

AD-A172 963

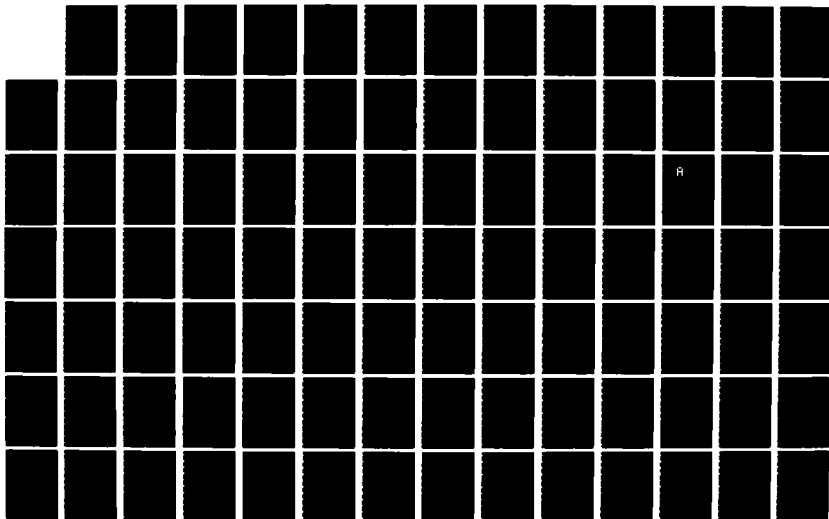
A LOW COST VIDEO DISPLAY SYSTEM USING THE MOTOROLA 6811 1/2
SINGLE-CHIP MICROCOMPUTER(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH K E WILLIAMS AUG 86

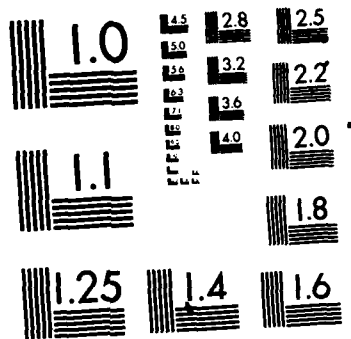
UNCLASSIFIED

AFIT/CI/NR-86-179T

F/G 9/2

NL






MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A172 963

DTIC FILE COPY

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/CI/NR 86- 179T	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Low Cost Video Display System Using The Motorola 6811 Single-Chip Microcomputer		5. TYPE OF REPORT & PERIOD COVERED THESIS/DISSEMINATION
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Kevin Eugene Williams		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AFIT STUDENT AT: The University of Texas		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 1986
		13. NUMBER OF PAGES 162
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASS
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES APPROVED FOR PUBLIC RELEASE: IAW AFR 190-1 <div style="text-align: right;">  LYNN E. WOLAVER 26 SEP 86 Dean for Research and Professional Development AFIT/NR </div>		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) ATTACHED ...		

DTIC
ELECTE
OCT 14 1986
S E D

A LOW COST VIDEO DISPLAY SYSTEM USING THE
MOTOROLA 6811 SINGLE-CHIP MICROCOMPUTER

by

KEVIN EUGENE WILLIAMS, B.S.E.E.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

August 1986

Acknowledgments

I would like to thank Dr. G. J. Lipovski for providing me with many ideas and suggestions throughout the course of this thesis.

July 18, 1986

Kevin E. Williams

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution	
Availability Codes	
Dist	
Special	
A-1	



TABLE OF CONTENTS

LIST OF FIGURES.....	viii
Chapter	
1: Introduction.....	1
1.1 Background.....	1
1.2 Purpose of Thesis.....	2
1.3 Organization of Thesis.....	3
 2: The Motorola 6811 Single-Chip Microcomputer.....	4
2.1 General Description.....	4
2.2 Register Set.....	4
2.3 Instruction Set.....	6
2.4 Addressing Modes.....	7
2.5 Operating Modes.....	7
2.6 Interrupts.....	8
2.7 On-Chip Peripheral Devices.....	8
2.8 Serial Peripheral Interface.....	9
 3: Keyboard Scan and Decoding.....	13
3.1 Keyboard Discussion.....	13
3.1.1 Debouncing.....	14
3.1.2 Key Identification.....	14
3.1.3 Generating the ASCII Code.....	14
3.1.4 Multiple Key Depression.....	15

3.2	Keyboard Hardware Design.....	15
3.2.1	Keyboard Hardware.....	16
3.2.2	Keyboard Hardware Operation.....	18
3.3	Keyboard Scan and Decoding.....	19
3.3.1	Keyboard Scan.....	20
3.3.2	Keyboard Decoding.....	23
4:	Video Signal Generation Using the M6811.....	26
4.1	CRT Theory and Timing.....	26
4.2	Video Generation Software Design.....	28
4.3	Screen Display Format Design.....	29
4.4	Synchronization Pulse Timing and Generation.....	31
4.5	Video Signal Generation.....	33
4.5.1	Discussion of Video Signal Generation Flowchart.....	36
4.6	Video/Keyboard Software Integration.....	46
4.7	Generating a Composite Video Signal.....	49
5:	The Modified BUFFALO Monitor.....	51
5.1	Determining the I/O Device.....	51
5.2	Data Input to BUFFALO.....	53
5.3	Data Output from BUFFALO.....	53
5.4	Summary of Changes to the BUFFALO Monitor.....	54
6:	Summary.....	56

Appendix A: A User's Guide to the Modified BUFFALO Monitor	57
A.1 System Startup.....	57
A.2 Available Memory for Programs.....	57
A.3 Interrupt Routines.....	58
A.4 Modified BUFFALO Commands.....	60
A.5 Debugging with the Modified BUFFALO Monitor.....	69
 Appendix B: Hardware Schematics	 70
 Appendix C: Using a Different Keyboard	 73
 Appendix D: Software Listing	 77
 Bibliography	 163

LIST OF FIGURES

Figure 2-1	M6811 Registers.....	5
Figure 2-2	SPI Master-Slave Relationship.....	10
Figure 2-3	SPI Associated Registers.....	12
Figure 3-1	Keyboard Hardware.....	17
Figure 3-2	Keyboard Scan and Decode Flowchart.....	21
Figure 4-1	Sample Font and Bit Pattern.....	31
Figure 4-2	Synchronization Pulse Timing.....	32
Figure 4-3	Even-Odd Frame Character Timing.....	34
Figure 4-4	Video Display Flowchart.....	38
Figure 4-5	Computing the Font Table Address.....	44
Figure 4-6	Keyboard/Video Subroutine Flowchart.....	48
Figure 4-7	Composite Video Circuit.....	50
Figure 5-1	Determining the I/O Device.....	52
Figure B-1	Keyboard Interface	70
Figure B-2	Composite Video Circuit.....	71
Figure B-3	RS-232 Circuit.....	71
Figure B-4	Possible Experiment Board Layout.....	72
Figure C-1	Radio Shack Keyboard Schematic.....	75

Chapter One

Introduction

1.1 Background

In 1972, Intel introduced the first central processing unit contained entirely on one chip, the 4004. [TANN84] Since then, continual improvements in semiconductor technology have enabled manufacturers to pack more and more components on a single chip. Now, in addition to the central processing unit, RAM, ROM, and peripheral devices can also be put on a single chip. This combination of central processing unit, RAM, ROM, and peripheral devices on a one chip is termed a single chip microcomputer [BAER80]. One of the most powerful single chip microcomputers available today is the Motorola 6811, (hereafter referred to as M6811).

The M6811 is the latest in the Motorola 68xx series of microprocessors-microcomputers. The M6811 features include an extended 6800 instruction set, 8K bytes of ROM, 512 bytes of EEPROM, 256 bytes of RAM, a 16-bit timer system, a serial communications interface (SCI), a serial peripheral interface (SPI), an eight channel analog-to-digital converter, and two 8-bit general purpose I/O ports. [MOT85A]

To help designers develop applications for the M6811, Motorola produced an evaluation module board. This board allows designers to develop hardware and software for an M6811 based system. A monitor called Bit Users Fast Friendly Aid to Logical Operation (BUFFALO) is resident in the 8K ROM of the board's M6811.

The monitor can be used via a terminal or host computer. BUFFALO also includes a debugger and a one line assembler/disassembler. [MOT85B]

Obviously the M6811 can be used in a variety of applications such as instrumentation and control systems. In the academic world, the M6811 can be used as a learning tool in an undergraduate level microprocessor class. However, developing an application using the M6811 requires the evaluation module board and a terminal as previously described. Consequently, acquiring the number of evaluation module boards and terminals to equip a microprocessor lab would be very costly. Also, students would be limited to working in the lab as very few students have the financial resources needed to acquire their own evaluation module board and terminal. In order to make the M6811 available for program development at a low cost, a different display method is needed.

Fortunately, as it will be shown in this thesis, the M6811 itself is capable of video signal generation. Indeed, the primary reason the M6811 was chosen for this project was because of its on-board SPI. It is the SPI which generates the high serial data rate necessary for a video display.

1.2 Purpose of Thesis

The purpose of this thesis was to design a low cost video display system using the M6811. The video display system accepts input from a keyboard and generates a composite video signal to display that data on an ordinary television set. The existing BUFFALO monitor was modified to work with this display system. The BUFFALO software along with the keyboard and video display software resides in the 8K ROM of the M6811. Students will use the available RAM and

EEPROM for developing their programs. Furthermore, the low cost video display system is easily transported and usable with an ordinary television set instead of an expensive terminal. Thus, a student working on an experiment can take the system home and use a regular television to work at home.

Low cost implies a minimal system and indeed one objective was to keep the number of components in the system to an absolute minimum. Thus, an external CRT controller, which would also require external memory, was not used to generate the video signals.

1.3 Organization of Thesis

The following is a brief overview of the organization of this thesis. In Chapter Two, a more detailed discussion of M6811 is given. Then, in Chapter Three the keyboard scanning and decoding algorithms along with the hardware are explained. Chapter Four discusses how the M6811 is used to generate the video signal. Chapter Five details how the modified BUFFALO monitor works with the keyscan/video display software and contains a summary of the modifications made to the original BUFFALO monitor. Chapter Six is a brief summary. Appendix A is a user's guide to the modified BUFFALO monitor. Appendix B contains the hardware schematics. Appendix C explains how to use a different keyboard with the system. Finally, Appendix D is a complete software listing. This software listing includes the modified BUFFALO monitor along with the keyscan/video display software.

Chapter Two

The Motorola 6811 Single-Chip Microcomputer

The purpose of this chapter is to explain the M6811 features and architecture. Only those features of the M6811 used by the display system will be covered in depth. The majority of the information in this chapter was extracted from [MOT85A].

2.1 General Description

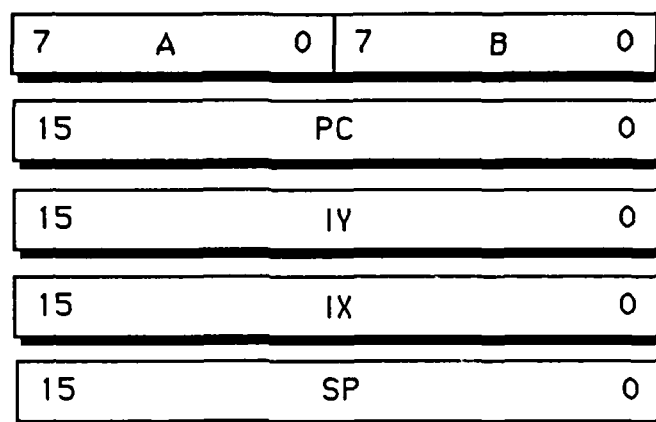
The M6811 is a single-chip microcomputer manufactured by Motorola, Inc. It provides on-chip memory of 8K bytes of ROM, 512 bytes of EEPROM and 256 bytes of static RAM. In addition, the following on-chip functions are provided:

- a) an 8 channel analog-to-digital converter
- b) a serial communications interface (SCI)
- c) a serial peripheral interface (SPI)
- d) a pulse accumulator
- e) an 8 bit bidirectional port
- f) an 8 bit output port
- g) a timer system which includes 3 input capture lines and 5 output compare lines

2.2 Register Set

The registers available to the programmer are shown in Figure 2-1. The following is a brief discussion of these registers:

Accumulators A and B: These are general purpose registers used to hold operands and results of arithmetic operations or data manipulations. These two accumulators can be concatenated in a single 16-bit accumulator called the D accumulator.



Condition Codes

S - Stop Disable

X - X Interrupt Mask

H - Half-Carry

I - I Interrupt Mask

N - Negative

Z - Zero

V - Overflow

C - Carry/Borrow

FIGURE 2 - 1. M6811 Registers

Index Registers IX and IY: These registers are used for indexed addressing. The 16-bit value in the index register is added to an 8-bit offset to compute

the effective address. The index registers can also be used as counters or temporary storage registers. The IY register operations require one more byte of machine code than the IX operations. Also, the IY operations require one more cycle of execution time.

Stack Pointer SP: This 16-bit register contains the address of the next free location on the stack. The stack operates on a last-in-first-out basis. When a byte is added (pushed) onto the stack, the byte is stored at address contained in the SP, then the SP is decremented. Conversely, when a byte is removed (pulled) from the stack, the SP is incremented, then the byte is retrieved from the address contained in the SP.

Program Counter PC: The program counter is a 16-bit register that contains the address of the next instruction to be executed.

Condition Code Register CCR: The condition code register is an 8-bit register in which each bit signifies the results of the instruction just executed. These bits can be individually tested by a program and a specific action can be taken as a result of the test.

2.3 Instruction Set

The M6811 instruction set is a superset of the M6800. There are 91 new opcodes available in the M6811. Among the enhancements of the M6811 instruction

set are addition of 16 by 16 divide instructions, STOP and WAIT instructions, and bit manipulation instructions.

2.4 Addressing Modes

The M6811 has six addressing modes. These modes are as follows:

- 1) Immediate - the actual operand is contained in the byte(s) immediately following the opcode.
- 2) Direct - the single byte following the opcode is assumed to be the least significant byte of the address of the operand. The most significant byte is assumed to be 00.
- 3) Extended - The two bytes following the opcode are the address of the operand.
- 4) Indexed - the X or Y register is used to calculate the effective address of the operand. An 8-bit unsigned offset is added to the index register.
- 5) Inherent - the operand is a register and is identified in the opcode.
- 6) Relative - used for branch instructions. If the branch condition is true, an 8-bit signed quantity is added to the program counter to form the effective branch address. If the branch condition is false, then the program continues with the next instruction.

2.5 Operating Modes

The M6811 can be operated either the single chip mode or the expanded multiplexed mode. In the expanded multiplexed mode, the M6811 can communicate with an external address and data bus. This allows the M6811 to access external

memory and peripheral devices. In the single chip mode, the external address and data buses are not used. For this thesis, the single chip mode will be used.

2.6 Interrupts

The M6811 supports both external and internal hardware interrupts. There are seventeen internal hardware interrupts and one software interrupt. A hardware priority circuit determines resolution of simultaneous interrupts. One of the interrupts may be elevated the highest priority in the resolution circuit.

When an unmasked interrupt occurs, a vector address for that interrupt is fetched from the internal ROM. All the registers are saved (except the SP) and control is passed to the vector address.

2.7 On-Chip Peripheral Devices

The following is a brief description of the on-chip peripheral devices on the M6811:

- 1) Parallel I/O: There are two ports available for general purpose I/O. Each bit in these ports may be configured as either input or output. One additional port is available for output only.
- 2) Serial Peripheral Interface: This device is used by keyboard decode/video signal generation software. Since it is the main device used for this thesis, it will be discussed in some detail in section 2.7.
- 3) Serial Communications Interface: This device provides full duplex, asynchronous communication capability. It may be set up in many

different configurations to allow it to communicate with a variety of systems.

- 4) Analog-to-Digital Converter: This device is an 8-channel multiplexed-input successive approximation A/D converter. The user connects a high and low reference to pins on the M6811. The digitized value of the input analog signal will be a value between 0 and \$FF. This value is ratiometric with respect to the high and low reference voltages.
- 5) Programmable Timer: The M6811 has a 16-bit free running counter, three input compare registers, and five output compare registers. The free running counter can be used by reading the most significant byte of the counter. The least significant byte is placed in a buffer. The input compare registers latch the value of the counter when an edge is detected on the associated input compare pin. The output compare registers compare the value stored in their associated register with the counter. The user specifies which actions are taken when a match occurs, such as generating an interrupt.

2.8 Serial Peripheral Interface

The SPI is the device used to read the keyboard and generate the signal to build a composite video signal. Subsequent chapters of this thesis will make reference to the SPI, so it is important to understand the details of this device. For an extremely detailed explanation of the SPI, see [GALL84].

The SPI allows full duplex, synchronous bit transfers at clock rates of up to 1.05 MHz. The SPI may be operated as a master or slave, but will only operate in

the master mode for this thesis. Figure 2-2 shows a block diagram of the SPI as a master connected to a slave. Later, it will be shown the "slave" is actually the hardware interface with the keyboard.

As Figure 2-2, shows there are three signals associated with the SPI. The first of these is the MOSI line. MOSI is short for Master Out - Slave In. There is a register in the SPI called the Serial Peripheral Data I/O Register (SPDR). When a

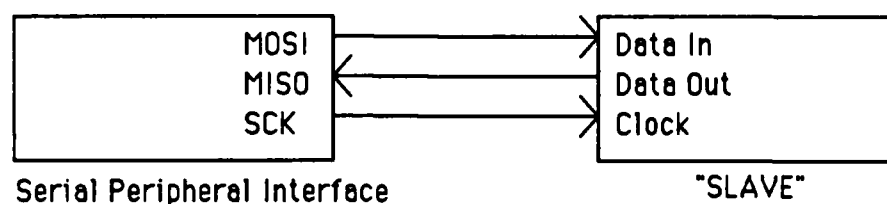


Figure 2 - 2. Master-Slave Relationship

byte of data is written to the SPDR the data is output on the MOSI line serially with the most significant bit sent first. The MISO, or Master In - Slave Out, is an input the SPI. It used by the slave to send a byte of data in serial form to the SPI. Thus, full duplex operation of the SPI is achieved via the MOSI and MISO lines. When a byte of data is written to the SPDR and shifted out on the MOSI line, the slave starts sending its byte of data on the MISO line. In essence, the master and slave exchange a byte of data. The synchronization for this transfer is the SCK line. This clock signal is activated when the byte is written to the SPDR and deactivated after the last bit of the byte has been sent.

There is another signal associated with the SPI that is not shown. It is called the Slave Select line. If the M6811 is used in a system in which several devices can

control the bus, the Slave Select line is used by the mode fault circuitry to detect bus conflicts. However, in this thesis, the M6811 is the only device that can ever be the "master". Therefore the mode fault circuit is not needed in this application and the Slave Select line is used as an output line.

Besides the SPDR, there are three other registers associated with the SPI system. They are the Serial Peripheral Control Register (SPCR), the Serial Peripheral Status Register (SPSR), and the Port D Data Direction Register (DDRD). Figure 2-3 illustrates these registers. The SPCR is used to initialize the SPI and set its operating parameters. The SPSR indicates the completion of data transfer between the master and slave. The SPIF bit of the SPSR is set to one upon completion of a transfer. No further transfers of data will be allowed until this bit is cleared. To clear the SPIF bit, the SPSR must be read followed by either a read from or write to the SPDR. The SPSR also tells if a write collision occurred or a mode fault was detected. A write collision is caused by trying to write to the SPDR while a transfer is taking place. Finally, the DDRD is used to connect the SPI signals to the port D pins as inputs or outputs. A one configures the pin as an output and a zero makes it an input. (Note the TxD and RxD bits are part of the SCI which also uses the port D pins.)

SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR2	SPCR
------	-----	------	------	------	------	------	------	------

SPIE - if set, SPI interrupts are allowed

SPE - if set, SPI system connected to external pins

DWOM - if set, SPI pins function as open drain outputs

MSTR - if set, SPI is the master device

CPOL, CPHA - determine polarity and phase of SCK

SPR1, SPR2 - select baud rate of SCK

SPIF	WCOL		MODF					SPSR
------	------	--	------	--	--	--	--	------

SPIF - set upon completion of data transfer

WCOL - if set, write collision occurred

MODF - if set, a multi-master conflict occurred

		\overline{SS}	SCK	MOSI	MISO	TxD	RxD	DDRD
--	--	-----------------	-----	------	------	-----	-----	------

Figure 2-3. SPI associated registers

Chapter Three

Keyboard Scan and Decoding

This purpose of this chapter is to describe how the M6811 is used to scan and decode a keyboard. The hardware used to interface the keyboard to the M6811 is described. The algorithm used to scan and decode the keyboard is presented.

3.1 Keyboard Discussion

A keyboard is simply a matrix of switches. When a single keyboard key is depressed, a unique row and column of the matrix are connected. Finding out which column and row are connected when a key is depressed is the heart of decoding a keyboard. How this decoding is done will be discussed shortly.

There are currently two types of keyboards, the encoded keyboard and the non-encoded keyboard. The encoded keyboard has the necessary hardware to decode the keyboard and supply the ASCII code of the depressed key. The non-encoded keyboard is nothing more than a matrix of rows and columns. All of the decoding must be done in software. The extra hardware required for the encoded keyboard makes it more expensive than the non-encoded keyboard. [ZAKS81] Because of its lower cost, a non-encoded keyboard will be used for this project.

Use of a non-encoded keyboard will require software to accomplish several tasks. These tasks include the following:

- Debouncing the keyboard.
- Identifying the row and column of the depressed key.

-- Generating the ASCII code of the depressed key.

-- Handling multiple-key depression.

3.1.1 Debouncing

Debouncing is required because there is an oscillation period ("bouncing") of 10 to 20 milliseconds before a switch contact stabilizes. [LIPO80] [ZAKS81] This happens when the switch transitions from open to closed and visa versa. To debounce the keyboard in software, a delay of 10 - 20 milliseconds is required. When a key has been depressed and detected by the software, the 10 - 20 millisecond delay is invoked. After the delay, the keyboard is checked again. If the same key is depressed, we have a valid key depression. In this system, the keyboard is sampled every 16.67 milliseconds. This 16.67 milliseconds corresponds to the time it takes to display one frame of video data. Thus, the keyboard is sampled once every frame during non-display periods. To get a valid key depression, the same key must be depressed for two successive samples.

3.1.2 Key Identification

To identify the depressed key, the row and column of the depressed key must be identified. In this system, a "walking zero" algorithm is used to identify the row and column. This algorithm is presented in detail in Section 3.3.

3.1.3 Generating the ASCII Code

There are a couple of different ways to generate the ASCII code for a depressed key. One way is to derive the code mathmatically. Generally, this will require a certain arrangement of the keyboard matrix. A second method is to use a

lookup table. In this method, the row and column of the depressed key are used to generate an offset into a lookup table which contains the ASCII codes. This lookup table method is used in this system for one important reason - using a lookup table allows a different keyboard to be used. Then only the lookup table need be modified when a different keyboard is used. If the mathematical method had been used, then the program section that computes the code would have to be changed when a different keyboard is used. With the program in the ROM section of the M6811, changing the program is not a viable option. Thus, the lookup table is the best option for ASCII code generation in this system.

3.1.4 Multiple Key Depression

The simplest way to deal with multiple key depression is to wait until one key is released before attempting to decode the keyboard again. In other words, once a key has been depressed and decoded don't try to decode the keyboard again until there are no keys being depressed. While this method would not be good for high speed typing, it is suitable for this system. This is because any input line to the monitor will usually consist of fewer than 10 keystrokes. This method also has the added benefit of requiring less software to implement.

3.2 Keyboard Hardware Design

The M6811 provides two different ways to design and implement the keyboard hardware. One way is to use two of the 8-bit parallel I/O ports. One of the ports would be connected to the columns of the keyboard matrix and the other would be connected to the rows. This method offers the advantage of simplicity in

terms of both hardware and software at the expense of tying up two of the M6811's 8-bit I/O ports. The second method is to use the SPI. Obviously, this method will require external hardware and more software. But it has the advantage of not tying up two 8-bit I/O ports. It is felt that keeping the two 8-bit I/O ports available for student projects is more important than using them for keyboard scanning. In addition, since it is already being used for video signal generation, using the SPI for the keyboard scanning optimizes its use. For these reasons the SPI will be used for the keyboard scanning along with some external hardware.

3.2.1 Keyboard Hardware

Refer to the keyboard hardware schematic, Figure 3-1, for the following discussion. The interface between the SPI and the keyboard matrix consists of a 74LS164 8-bit parallel-out serial shift register and a 74LS165 8-bit parallel-load shift register. The 74LS164 outputs are connected to the columns of the keyboard matrix, while the parallel inputs to the 74LS165 are connected to the rows of the keyboard matrix. The MOSI output from the SPI is connected to the serial input A of the 74LS164 and the serial output of the 74LS165 is input to the MISO pin of the SPI. The SCK output from the SPI is connected to the clock input for both chips. Notice the master-slave relationship between the keyboard hardware and the SPI. There are pullup resistors connected to the parallel inputs of the 74LS165. This guarantees a logic "1" is presented to the parallel inputs if no key is depressed on the keyboard. Note the SS line of the SPI is connected to the load/shift input of the 74LS165. This line will be used to tell the 74LS165 when to latch the data present on the rows of the keyboard.

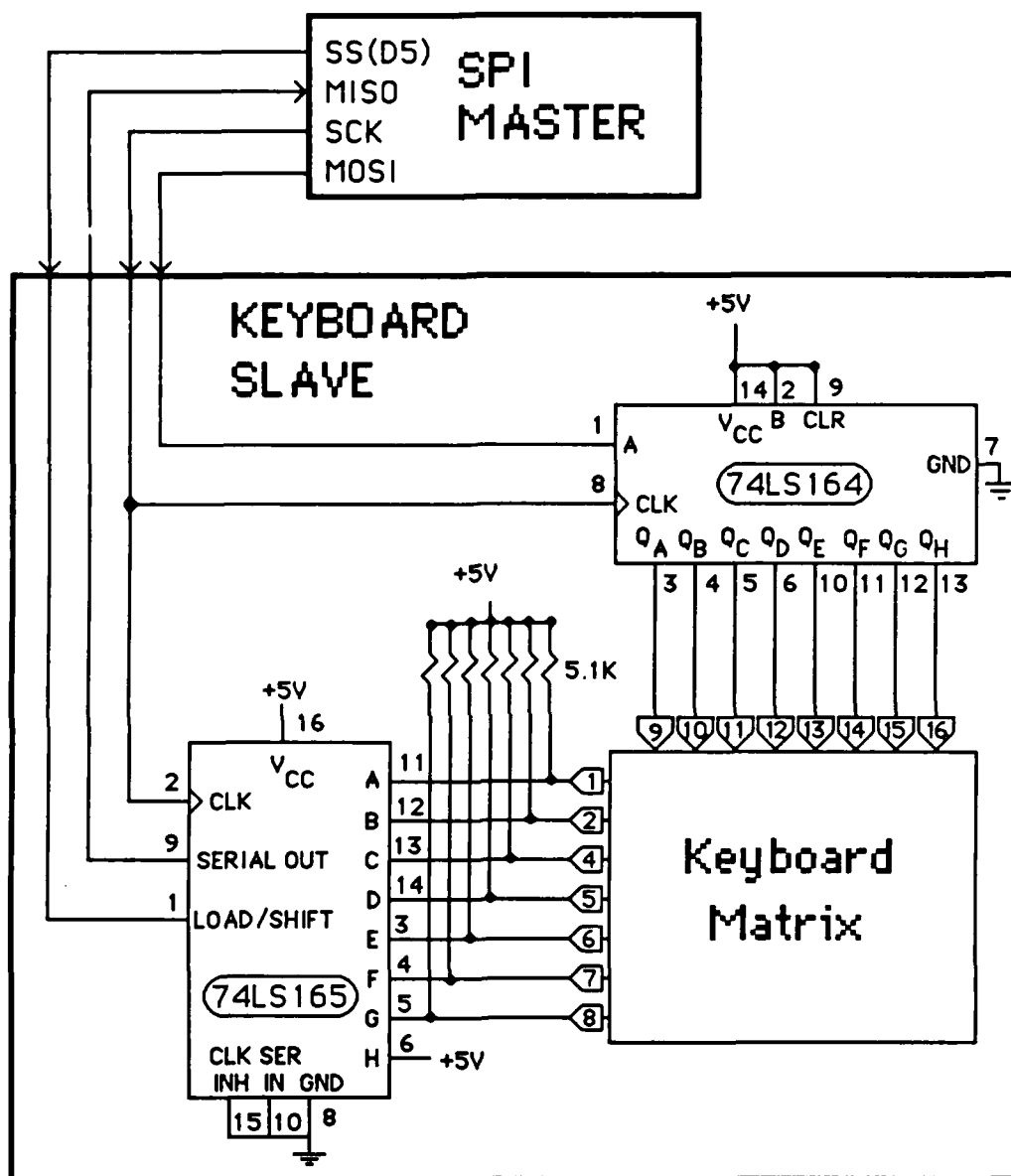


Figure 3-1 Keyboard Hardware

It was pointed out earlier that the SS line could be used as a general purpose output if the mode fault circuit was not needed. Since the mode fault circuit is not needed (because there can only be one master in this system) the SS line is used as an output. Hereafter, this line will be referred to as the D5 line (pin 5 of port D) because it is functioning as a general purpose output line, not as a slave select input line. The keyboard is from a Radio Shack Color Computer. This keyboard has an 8 column by 7 row matrix. Since there are only 7 rows, the H input to the 74LS165 is tied high to make sure the walking zero algorithm works properly. The hardware and software can easily be modified to use a different keyboard and this is described in Appendix C.

3.2.2 Keyboard Hardware Operation

The following is a discussion of how the keyboard hardware is designed to operate. The software writes a one byte pattern to the SPDR. The action of writing the pattern to the SPDR initiates the serial transfer of the byte on the MOSI line. The SCK synchronizes the transfer with the 74LS164. Upon completion of the transfer, the pattern is now present on the columns of the keyboard. This is how the patterns are set up on the columns of the keyboard. Recall that the SPI also receives a byte of data on its MISO line. We have already seen the MISO line is connected to the serial output of the 74LS165. So at the same time the bit pattern is sent to the 74LS164 for the column pattern, a byte representing the row pattern is received from the 74LS165. The row pattern must be loaded into the 74LS165 shift register by sending an active low load pulse to the 74LS165. This is done in software by pulsing the D5 line. Thus, the hardware allows the keyboard to be scanned by

sending column patterns to and receiving row patterns from the keyboard hardware interface.

3.3 Keyboard Scan and Decoding

This section presents the algorithm used to scan and decode the keyboard. The software program listing is located in Appendix D. The algorithm uses what is known as a "row scanning" technique. [LIPO80] [ZAKS81] In row scanning, a pattern is presented on the columns. The first pattern consists of a zero in the least significant bit (bit 0) and ones in all the other bits. After this pattern is presented on the columns, the row pattern is latched and read. If the value of the row pattern is \$FF (all ones), then no key along the least significant bit's column has been depressed. The next pattern is sent to the 74LS164. However, this pattern is different. This zero is now shifted into bit 1 with all other bits set to one. The row pattern is read and again checked against \$FF. If it is not \$FF, then a key has been depressed in the bit 1 column. The column of the depressed key is known and the row can be found from the row pattern. With this information, the corresponding ASCII value is found in a lookup table. Notice that if no key is depressed in the bit 1 column, then the zero would be shifted into bit 2 and the process would repeat until bit 7 is reached. Thus, we have a "walking zero" algorithm as the zero in the column pattern walks through the byte. Once the bit 7 pattern has been sent, all the columns have been scanned. If no key has been depressed, then the keyboard is not scanned again for another 16.67 milliseconds. The 16.67 milliseconds corresponds to time it takes for display of one frame of a video signal.

3.3.1 Keyboard Scan

The flowchart for the keyboard scan is shown in Figure 3-2. At the start of the keyboard scan the variables COLCNT and ROWCNT are initialized to zero. These two variables are used to hold the column count and row count respectively. Then accumulator A is initialized with the first column pattern value of \$FE (1111 1110). This first pattern is sent and at the same time a meaningless byte is received from the 74LS165. The 74LS165 data inputs are latched by pulsing the load line of the 74LS165 via the D5 pin of the M6811. The 74LS165 now has a valid row pattern latched into it. The carry bit is set and a ROLA instruction is executed. Accumulator A now has the value \$FC (1111 1101). This second column pattern is sent at the same time the first row pattern is received. The received row pattern is compared with \$FF to determine if a key is depressed. If a key is depressed, then the row pattern value will not be equal to \$FF and control will jump to the decode routine. If no key is depressed, then the value of COLCNT will be incremented by one. This is to keep track of the value of the column being examined for key depression so it can be decoded later. Now, the carry bit is set and a ROLA is executed. The carry bit is then checked to see if it is zero. This is done to see if all the column patterns have been sent. If the carry bit is not zero, then the program will loop back as shown in Figure 3-2 to repeat the process. Eventually, after the eight column patterns have been sent, the carry bit will be zero. There is still one row pattern to retrieve from the 74LS165. This is done by sending a load pulse to the 74LS165 and then writing an arbitrary byte to the SPDR. The value of the byte sent this time is unimportant because we are not concerned with setting up a valid pattern

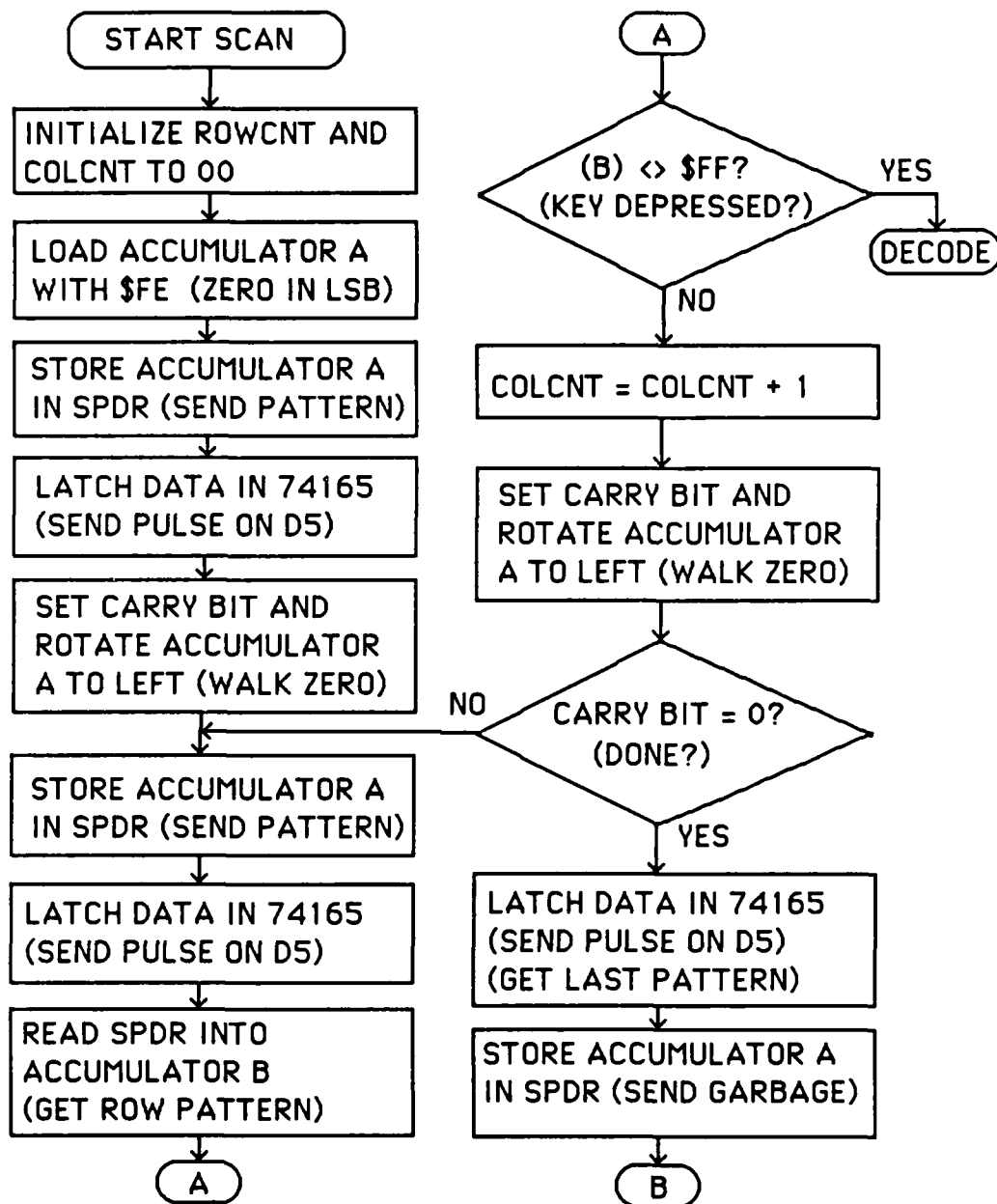


Figure 3-2. Keyboard Scan and Decode Flowchart

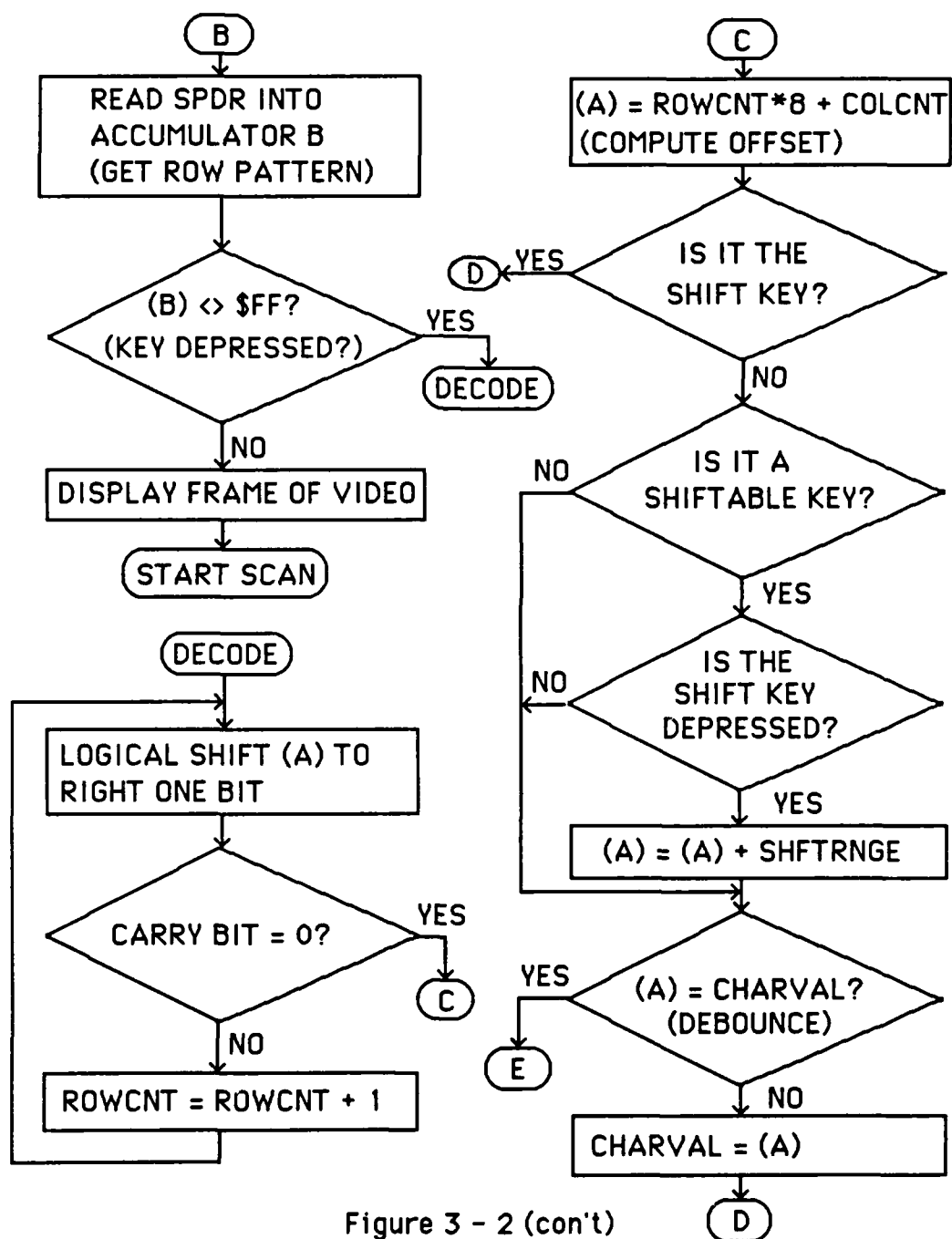


Figure 3 - 2 (con't)

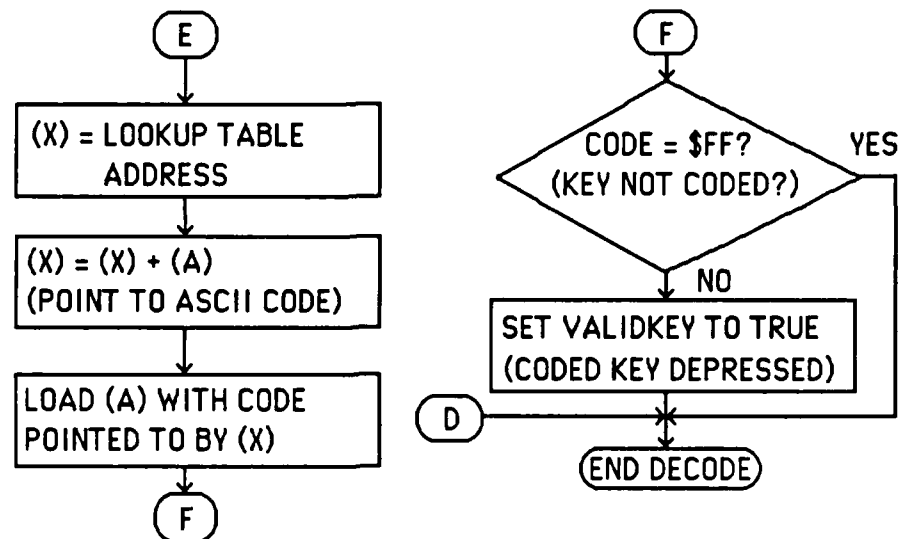


Figure 3 - 2 (con't)

on the columns but rather only interested in getting the last row pattern from the 74LS165. The last row pattern is checked for a key depression. If there is no key depression the decode routine is skipped; if there is a key depression, then the decode routine is executed.

3.3.2 Keyboard Decoding

The decoding flowchart is also depicted in Figure 3-2. In order for the decode routine to be entered, a key depression must have been detected by the scan routine. At this point, the column number of the depressed key is contained in COLCNT, and the row pattern is contained in accumulator B. The first thing to do is translate the row pattern into a row number. This is accomplished by executing a LSRB and checking the carry bit for a zero. If the carry bit is not zero, then the value of ROWCNT is incremented by one and the loop is repeated until the carry bit

is zero. Once the zero is reached, the correct row number value is in ROWCNT.

The value of ROWCNT is multiplied by 8 and added to the value of COLCNT. This value is first checked to see if it is the shift key. If it is, then only the shift key is down. This is because the shift key is the last key on the matrix; it has the highest value for $8 * \text{ROWCNT} + \text{COLCNT}$. If it is not the shift key, then the value is checked to see if it is a shiftable key. If the value falls in the shiftable range, then the shift key is checked to see if it is depressed. This is done by sending a pattern to the 74164 which contains a zero in the column the shift key is located. The row pattern is latched and retrieved. This pattern is checked to see if there is a zero in the row the shift key is located. If the shift key is depressed, then the constant SHFTRNGE is added to the computed value. This new value is now compared with the value of CHARVAL. CHARVAL contains the computed value of the last decoded key. The purpose of this comparison is to debounce the keyboard by requiring the same value for a decoded key on two successive keyboard scans 16.67 milliseconds apart. If the comparison fails, then the computed value becomes the new value of CHARVAL and the decode routine is exited.

If there is a match between the computed value and CHARVAL, then the only computation left is to load index register X with the value of the starting address of the lookup table and add to it the computed offset. The ASCII code is loaded into accumulator A. Finally, the code is checked against \$FF. The value \$FF is used in the lookup table to indicate an unimplemented key. If the value of the code is not \$FF, then the flag VALIDKEY is set to true to indicate a valid key has been

depressed and properly decoded. At this point, program control returns to the caller.

The caller now has the ASCII code of the depressed key in accumulator A.

Chapter Four

Video Signal Generation Using the M6811

This chapter explains how the M6811 is used to generate the signals necessary for a composite video signal. First, there is a brief introduction to cathode ray tube (CRT) signal timing requirements. This is followed by a presentation of the algorithm used to generate the video and synchronization signals. The integration of the keyboard software with the video software is also described.

4.1 CRT Theory and Timing

The purpose of this section is to give a brief introduction to CRT theory and the signal timing requirements for the CRT's found in unmodified black-and-white television sets. For a more detailed explanation of CRT theory and timing, the reader may consult [KANE80], [LANC78], and [HERR72]

A CRT consists of an evacuated glass tube with a fluorescent coating on the inner surface of its rectangular screen and an electron gun positioned opposite the screen. The electron gun emits an electron beam which strikes the fluorescent screen producing a phosphor dot on the screen. The point where the electron beam strikes the screen is controlled by electromagnetically deflecting the electron beam. The intensity of the electron beam can also be controlled. This allows various shades of grey to be displayed on a black-and-white television.

The internal circuitry of a television automatically moves the electron beam in a set pattern or scan. In raster-scan televisions, the electron beam starts in the upper left corner of the screen. The beam moves to the right until it reaches the right side of

of the screen. When it reaches this point, the beam quickly moves back to the left side of the screen. This horizontal movement of the beam back to the right side of the screen is called horizontal retrace. The beam has also moved down very slightly in the vertical plane. This process continues until the beam reaches the lower right corner of the screen. After reaching the lower right corner of the screen the beam returns, called vertical retrace, to the upper right corner and the same process continues.

The frequency of the horizontal and vertical scans is preset. In the United States, the horizontal frequency is 15,750 Hz. In other words, the electron beam makes 15,750 traversals, or lines, across the screen every second. The vertical frequency is 60 Hz. Thus, the electron beam goes from top to bottom in $1/60$ th of a second (called a frame). This means there are 262.5 lines ($15750+60$) displayed as the electron beam moves from top to bottom 60 times every second. Unfortunately, not all of these 262.5 lines are usable for display. It takes about 20 horizontal lines of time to accomplish the vertical retrace. In addition, unless the television is of high quality, some of the horizontal lines will be either partially or completely out of view near the top and bottom of the screen.

The CRT requires three signals in order to produce a picture. These three signals consist of the vertical synchronization pulses, the horizontal synchronization pulses, and the video signal. The vertical and horizontal synchronization pulses do not drive the vertical and horizontal scans, but rather they synchronize with the CRT circuitry. This allows some variation in the signals provided to the television. This means the horizontal synchronization pulse, called HSYNC, can be varied somewhat but should be maintained within +5 to -10% of the 15,750 Hz frequency.

[KANE80] The HSYNC pulse width is typically between 5 -10 μ s. The vertical synchronization pulse, called VSYNC, requires a much higher accuracy. If the VSYNC pulse is not maintained within .01% of 60 Hz, a hum bar will cross the screen about every 10 seconds. [LANC78] The VSYNC pulse width is typically 3 horizontal lines long (about 200 μ s). Finally, the video signal controls the intensity of the CRT's electron beam. The higher the voltage of the video signal, the brighter the phosphor spot the electron beam produces. When all three of these signals are combined into a single signal, the single signal is called a composite video signal.

4.2 Video Generation Software Design

There are two possible directions to take in the design of the video generation software. One is an interrupt-driven design. In this design, the video generation software interrupts the BUFFALO monitor every 1/60 th of a second to display a frame of video data. This design uses one of the on-board counters to generate an interrupt when it is time to display a frame of video. In addition, external hardware is required to generate the synchronization signals. This design was attempted but proved unsuccessful because of difficulties synchronizing the timing. A second design was made which did not use interrupts. This design makes the video generation software a subroutine. The subroutine is called by the BUFFALO monitor whenever it needs data from the user or needs to display data for the user. This design does not need to use one of the on-board counters and does not need external hardware to generate the synchronziation signals. The subroutine design does not suffer from the timing synchronization problems encountered with the interrupt-driven design. This is because once the subroutine is invoked it starts the

timing from that point. Then it is merely a matter of counting machine cycles to ensure the timing is done properly. When the BUFFALO monitor has program control, the CRT screen will be blank. However, the BUFFALO monitor processes most commands so quickly that the user does not even notice the CRT going blank. The remainder of this chapter will explain the aspects of this design and how it works.

4.3 Screen Display Format Design

The screen display format consists of 5 character rows with 6 characters per row. This provides a 30 character display - more than enough for any possible input to the BUFFALO monitor. This format was not arbitrarily chosen. It was driven by the interaction of three factors: the speed of serial transfer from the SPI, the time required for the software to lookup the font patterns, and the use of as little of the on-board RAM as possible.

The amount of data that can be displayed on a CRT screen is directly related to the speed at which a serial bit stream can be generated. The serial bit stream consists of the "1" s and "0" s to be displayed on the screen. With a 2 MHz E clock, the maximum rate of serial transfer from the SPI is 1 MHz. A 1 MHz serial bit rate is not fast enough to provide anything close to a high resolution display. Recall that there are 15750 horizontal lines and 60 frames of data displayed every second. This results in one line of data displayed in 63.5 μ s, of which 5-10 μ s is used for the horizontal retrace and about 10 μ s is spent in a "non-viewable" area. This leaves about 40 μ s for display of video data. If each character takes 8 bits to display horizontally, only 5 characters could be displayed on a line. In this system, a

technique is used to make the effective width only 6 bits wide. (see section 4.5) Therefore 6 characters can be displayed in 36 μ s.

To determine how many rows of 6 characters can be displayed recall there are about 260 horizontal lines displayed every 16.67 ms. However, some of these lines are used during vertical retrace and others are not displayed on viewable portion of the screen. In this system, a total of 160 horizontal lines are used for display of data. Using a 5 by 7 font one might think 20 character rows ($160 \div 8$) could be displayed. However, it is impractical to display 20 rows of 6 characters in this system for three reasons. First, it would require 120 bytes, almost half the on-board RAM, be set aside for a character buffer. Secondly, there are not enough machine cycles in a horizontal line display time to enable the software to lookup all the display patterns. (This reason in itself rules out a 20 by 6 display) Finally, there is no need for a display this size when working with the BUFFALO monitor. The number of character rows for display has been set at 5. Each character row consists of 32 horizontal lines. Each scanline of a character is displayed for 4 horizontal lines. This scheme also allows enough time for the software to do all the font pattern lookups.

Each character of the font consists of 8 bytes. The font for the character "A" along with its corresponding bit pattern is shown in Figure 4-1. A "1" displays a white dot on the CRT while a "0" results in a black dot.

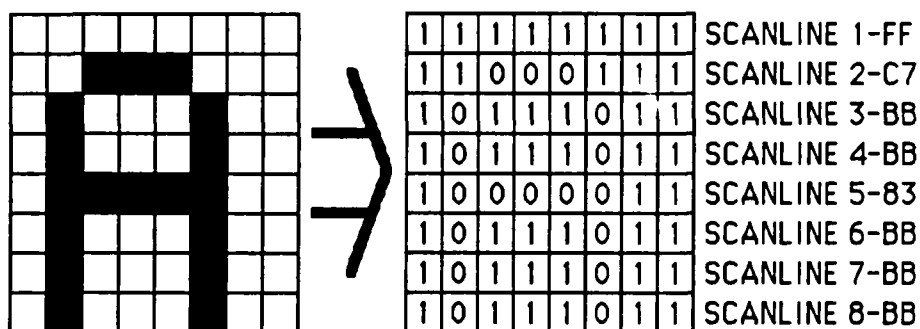
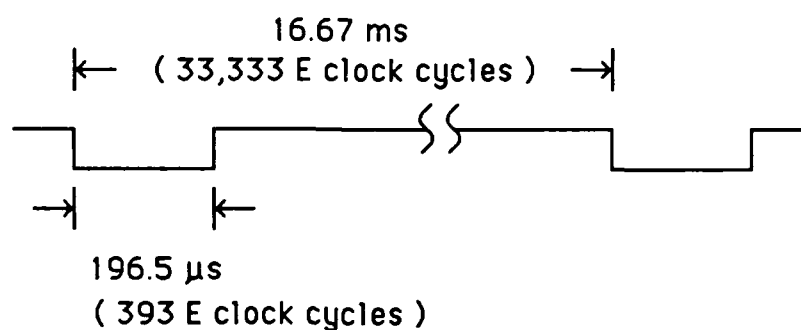


Figure 4 - 1. Sample Font and Bit Pattern

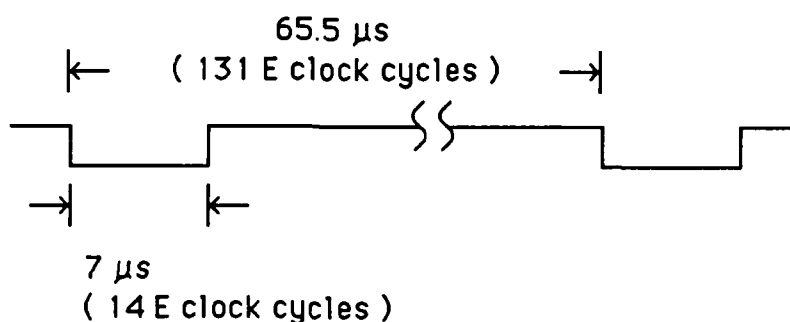
4.4 Synchronization Pulse Timing and Generation

This section explains the exact timing for the vertical and horizontal synchronization pulses used in the system. All values for the E clock cycles are based on a 2 MHz E clock.

The VSYNC pulse timing is shown in Figure 4-2a. The pulse width is 196.5 μ s and the frequency is 60 Hz. The VSYNC pulse width corresponds to exactly three horizontal line times. From the start of one frame to the start of the next takes 16.67 ms which equates to 33,333 E clock cycles. The HSYNC pulse timing is shown in Figure 4-2b. The pulse width is 7 μ s and the frequency is 1/65.5 μ s or 15,267 Hz. The HSYNC pulse frequency is slightly different from the 15,750 Hz standard but it is still within timing tolerances. This slower frequency is a result of needing more clock cycles per horizontal line in order to accomplish all the



(a) VSYNC pulse timing



(b) HSYNC pulse timing

Figure 4 - 2. Synchronization Pulse Timing

necessary tasks in the display software. A side effect of the slower HSYNC frequency is a reduction in the number of horizontal lines per frame. Instead of 262.5 lines per frame, there are now $15267/60$ or 254.5 lines per frame.

The generation of the VSYNC and HSYNC pulses is quite simple. Both VSYNC and HSYNC pulses are output on the D5 pin. This results in a composite synchronization pulse signal. When it time for a pulse to be output, a "0" is written to bit 5 of port D. After an appropriate delay for the pulse width a "1" is written to

bit 5 of port D. As stated earlier, the video generation software is called as a subroutine. The subroutine begins by outputting a VSYNC pulse. All synchronization timing starts from this point and HSYNC and VSYNC pulses are output at the proper times.

4.5 Video Signal Generation

This section explains the algorithm used to generate the video signal. The characters are represented in a 30-byte character buffer by their ASCII code (the characters are placed in the buffer by the BUFFALO monitor which will be discussed in Chapter 5). A 6-byte display buffer contains the font patterns to be displayed. There are two problems to solve in order to translate the ASCII codes in the character buffer into a proper video data signal. First, the patterns corresponding to ASCII codes in the character buffer have to be looked up in the font table and placed in the display buffer. Second, the patterns have to be displayed at exactly the right time every frame. These two problems will have to be solved in most any display system design. However, in addition to these problems, another problem surfaced while designing the solution for the M6811.

This additional problem was the result of the slow speed of the SPI. Because the SPSR has to be read after completion of a byte transfer, patterns cannot be continually sent to the SPDR for transfer. A pattern must be sent, the SPSR must be read after the transfer is complete, then another pattern may be sent. Not only does this reduce the number of characters that can be displayed on a horizontal line, but it also results in large gaps between characters on the CRT screen. This means only 3 or 4 characters could be displayed on the screen and these characters would

have significant gaps between them. To solve this problem a unique solution was devised.

The solution involves displaying half the characters in the character buffer during one frame followed by the other half of the characters the next frame. Figure 4-3 illustrates the timing relationships between the even and odd frame horizontal lines. The address of CHARBUFF is considered the first "even" character address, CHARBUFF+2 is the address of the second "even" character, etc. Likewise, CHARBUFF+1 is the first "odd" character address, CHARBUFF+3 is

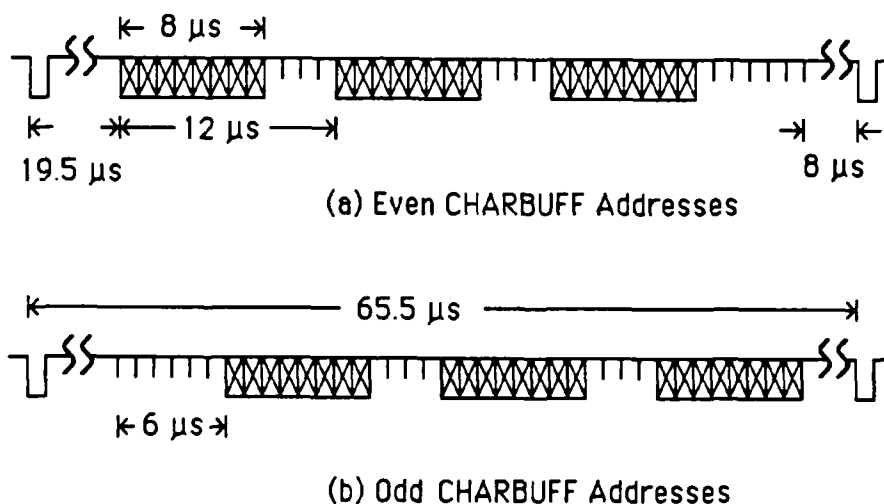


Figure 4 - 3. Even-Odd Frame Character Timing

the second "odd" character address, and so on. Three character patterns are displayed during each line with 12 μ s spacing separating the start of each character pattern. The 1 MHz SPI speed means each pattern takes 8 μ s of time to display. The timing of the frames is the same except that during an "odd" frame the character patterns are delayed by 6 μ s. This 6 μ s difference between the two frames is

accomplished by checking the variable FIELD. Based on the value of FIELD, 6 μ s of code is executed either before or after the active display time. There is one drawback of this scheme. Because the effective frame rate is reduced to 30 Hz, there is a slight but perceptible flicker of the display. However, the increased effective bandwidth achieved with this scheme outweighs the slight flicker of the display.

Also note the effective width of the characters is now only 6 bits. This is because of the overlap of the two frames. The last two bits of "even" characters occupy the same location as the first two bits of the "odd" characters. This makes the space between two characters on the screen appear to be only one bit wide.

This solution is integrated with the solution to two previously stated problems and the resulting algorithm is detailed in by flowchart in Figure 4-4. Before beginning the discussion of the flowchart it is important to make a point about its structure. One of the more difficult tasks in designing this section of the software was getting all the tasks accomplished in the available time per horizontal line. Also, it was a goal to keep the overall size of the program to a minimum because of the limited amount of ROM. To keep program size down meant designing repetitive loops wherever possible under constraint of the limited machine cycles available per horizontal line. Add to all of this the complication of generating "even and odd" frames and the problem is not trivial. The reader may wonder why the video generation software (see Appendix D) looks disjointed or instructions seem "out-of-place". This is because of all of these requirements and constraints. While the software may look disjointed and instructions may appear "out-of-place", it has to be in order to take advantage of every machine cycle.

4.5.1 Discussion of Video Signal Generation Flowchart

Refer to Figure 4-4 for the following discussion. The following is a list of the symbolic names used in the flowchart and their functions:

<u>SYMBOLIC NAME</u>	<u>FUNCTION</u>
CHARBUFF	Address of first character in the 30-byte character buffer
DISPBUF	Address of first display pattern in the 6-byte display buffer
FONTABLE	Address of first pattern in the font table
READBUF	Used to save address of first character of each character row
SCANLINE	Used to hold value of current scanline
SLCNT	Used to control number of times the same scanline is displayed
DISPTR	Holds address of next position in display buffer to be filled
FIELD	Used to indicate whether odd or even frame is displayed

At the start of every display frame, the first task accomplished is to toggle the FIELD variable. This ensures the display alternates between odd and even frame displays. Then the variables used by the routine are initialized. In addition, the first three locations of the display buffer are initialized to \$FF. Note that this is permitted because the first scan line of every character is \$FF. Index register Y and READBUF are initialized to point to CHARBUFF-1+FIELD. Thus, FIELD is used to point Y at either the "odd" or "even" character addresses. The (-1) is necessary because Y is incremented by one during the HSYNC pulse. DISPTR is

initialized to `DISPBUFF+2` as is index register `X`. The reason for this will become apparent shortly.

Next, a repetitive loop is entered. This loop will execute until `SLCNT` reaches 0. Since `SLCNT` is initialized to 3, it will execute three times. After the `HSYNC` pulse is output and index register `Y` is incremented, `FIELD` is checked. If `FIELD` is "1" (odd frame), then a 6 μ s section of code is executed. This 6 μ s of code consists of looking up a pattern pointed to by index register `X` and placing it in the display buffer address pointed to by `DISPTR`. Note that the first time through the loop `DISPTR` and index register `X` are pointing at the same location - `DISPBUFF+2`. So the first time through the loop `DISPBUFF+2` is written back to itself. This will not be the case on subsequent passes through the loop as index register `X` will then point to a new pattern in the font table. Next, an ASCII code is retrieved from the character buffer address pointed at by index register `Y`. The first time through the loop the content of either `CHARBUFF` or `CHARBUFF+1` is retrieved based on the value of `FIELD`. The value of `DISPTR` is incremented to point at `DISPBUFF+3`. It is now time to send the first pattern, located in `DISPBUFF`, via the SPI by writing to the `SPDR`. While this transmission is taking place, the ASCII code that was just looked up is multiplied by 8. Now it is time to send the second pattern, located in `DISPBUFF+1`. During this transmission, the value of `SCANLINE` is added to the value of `FONTABLE-$100` and this is added to `ASCII*8`. The reason for adding `FONTABLE-$100` is because the first entry in the font table is a space character, ASCII code \$20. Since $8*\$20 = \100 , \$100 must be subtracted from `FONTABLE` to get the correct offset. Figure 4-5 shows computation of the font table address graphically.

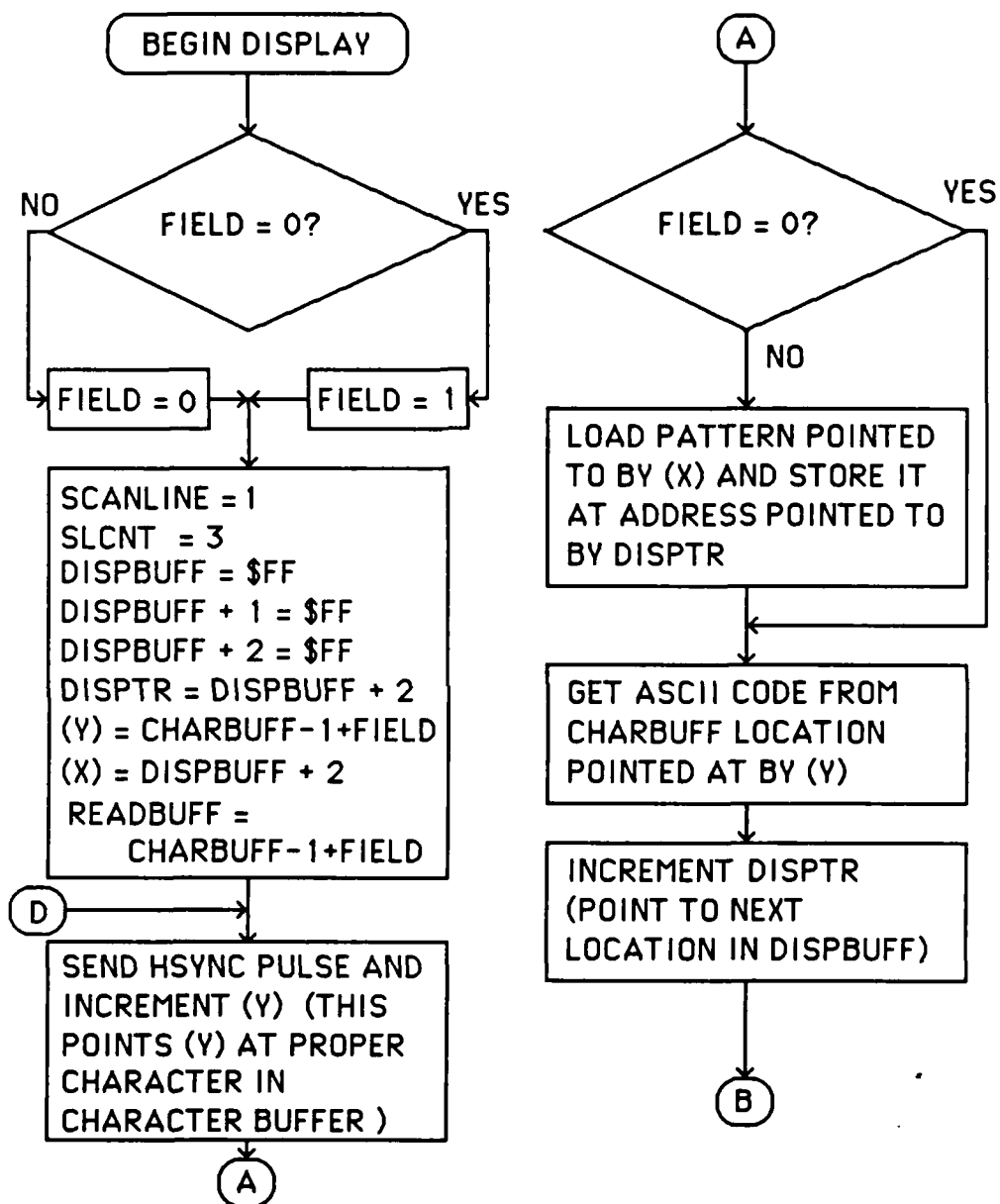


Figure 4-4. Video Display Flowchart

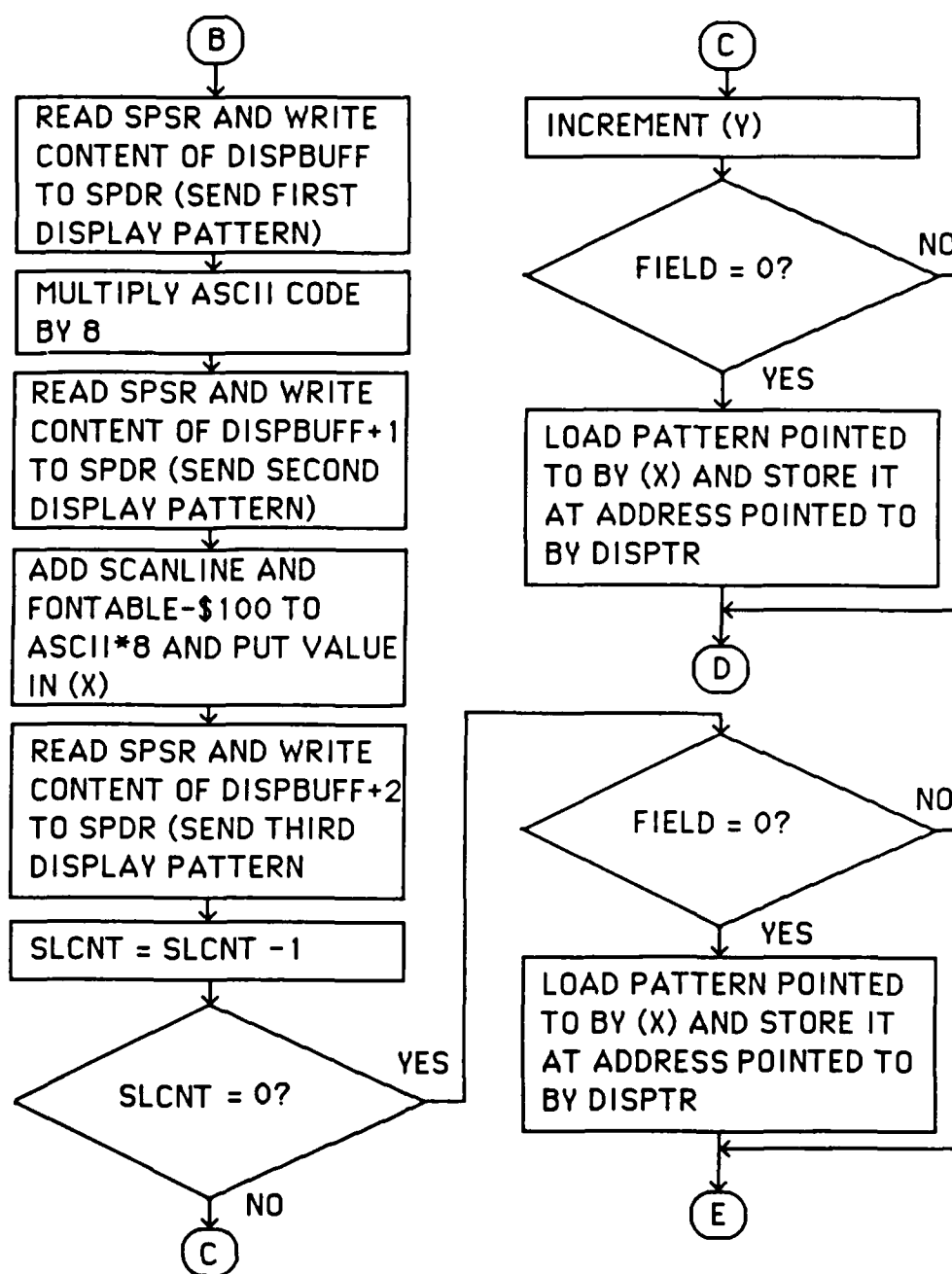


Figure 4-4. (Cont'd)

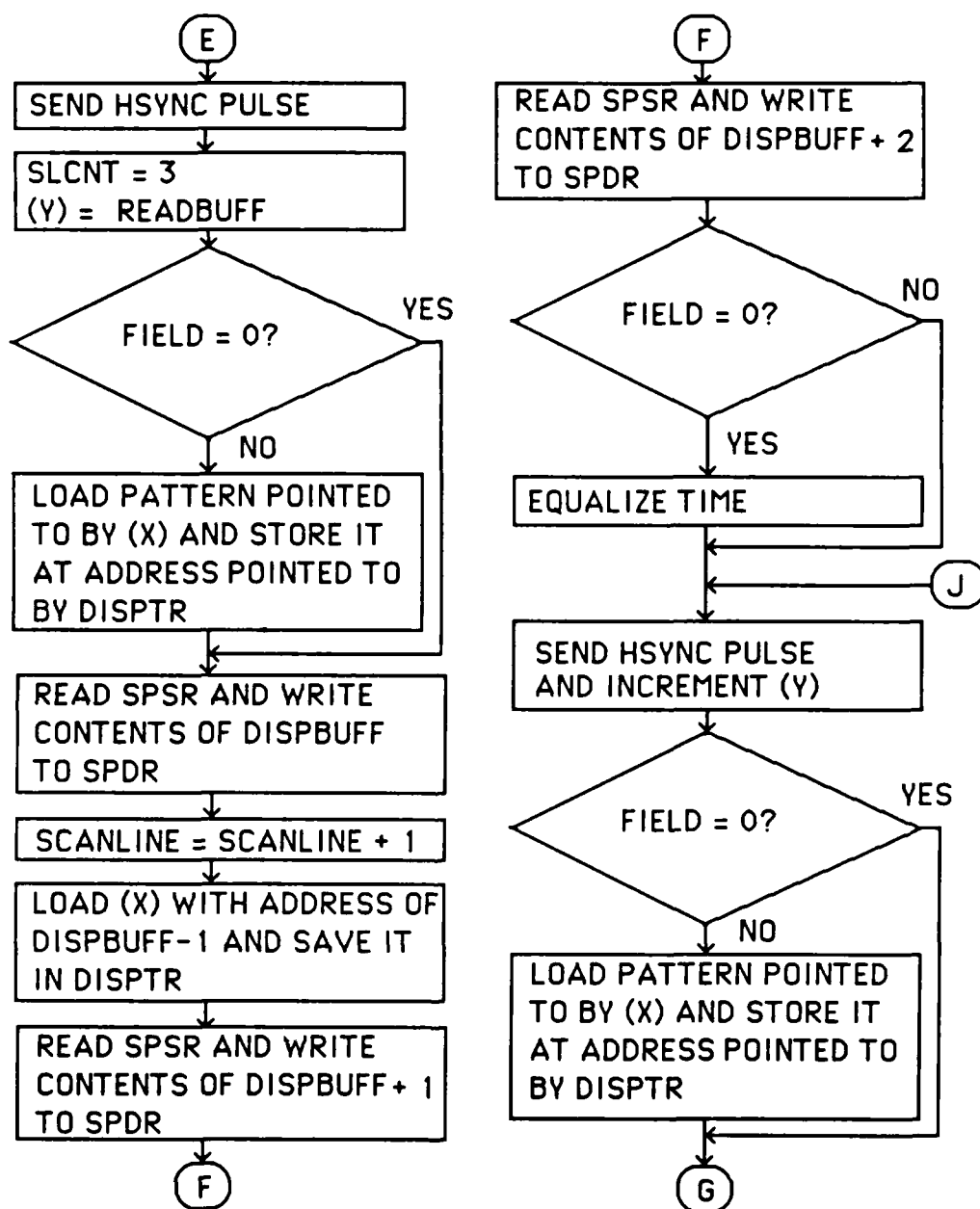


Figure 4-4. (Cont'd)

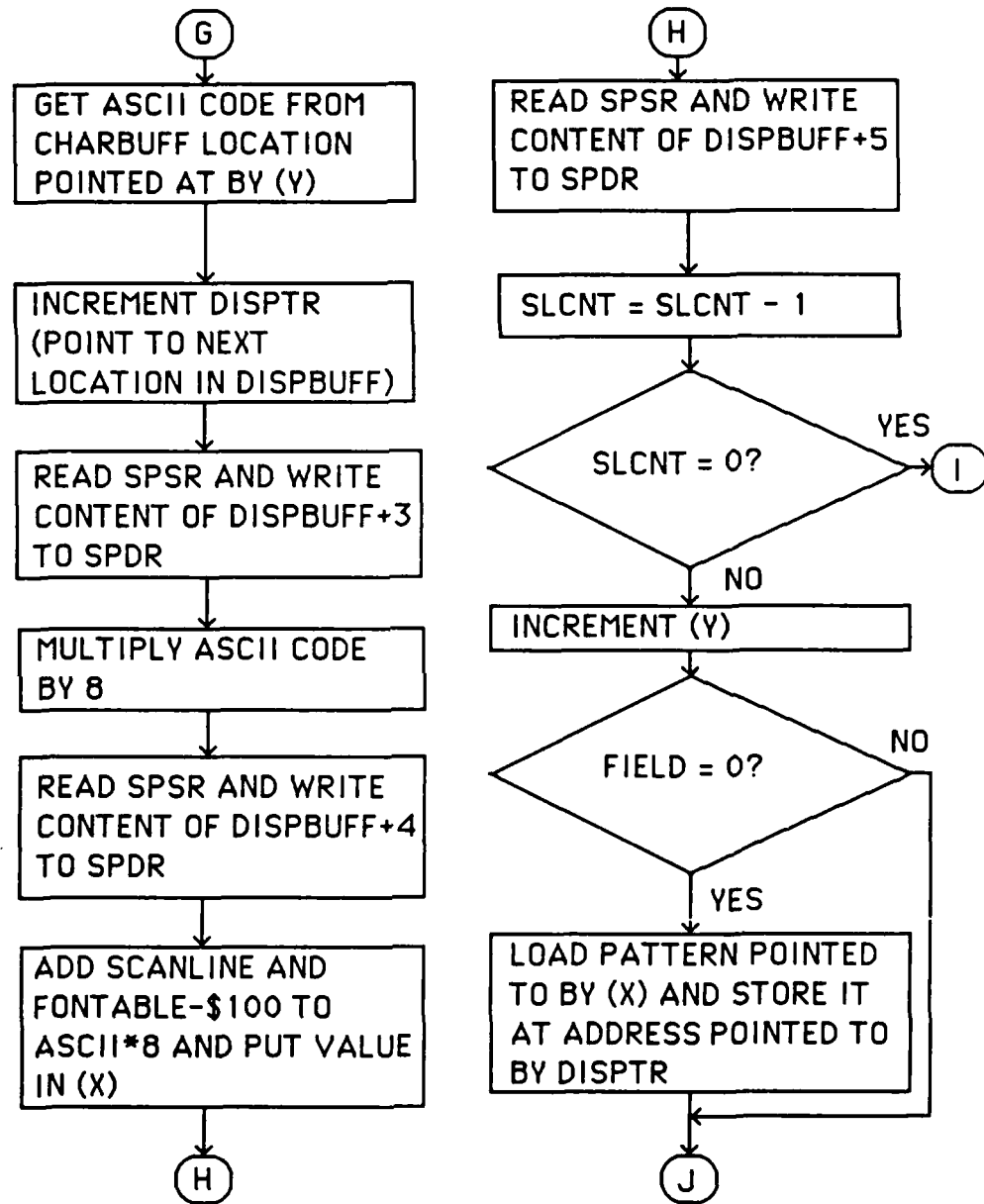


Figure 4-4. (Cont'd)

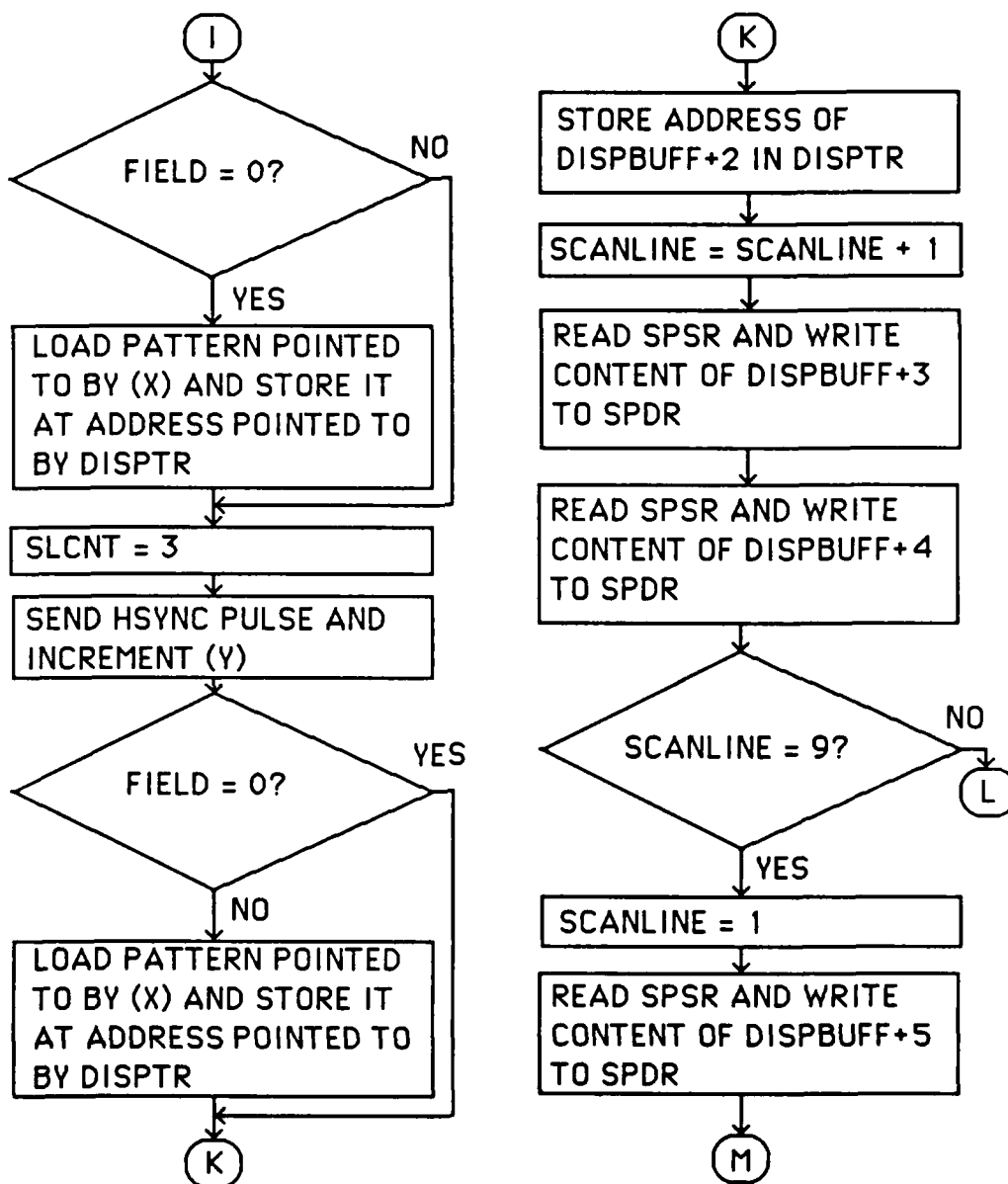


Figure 4-4. (Cont'd)

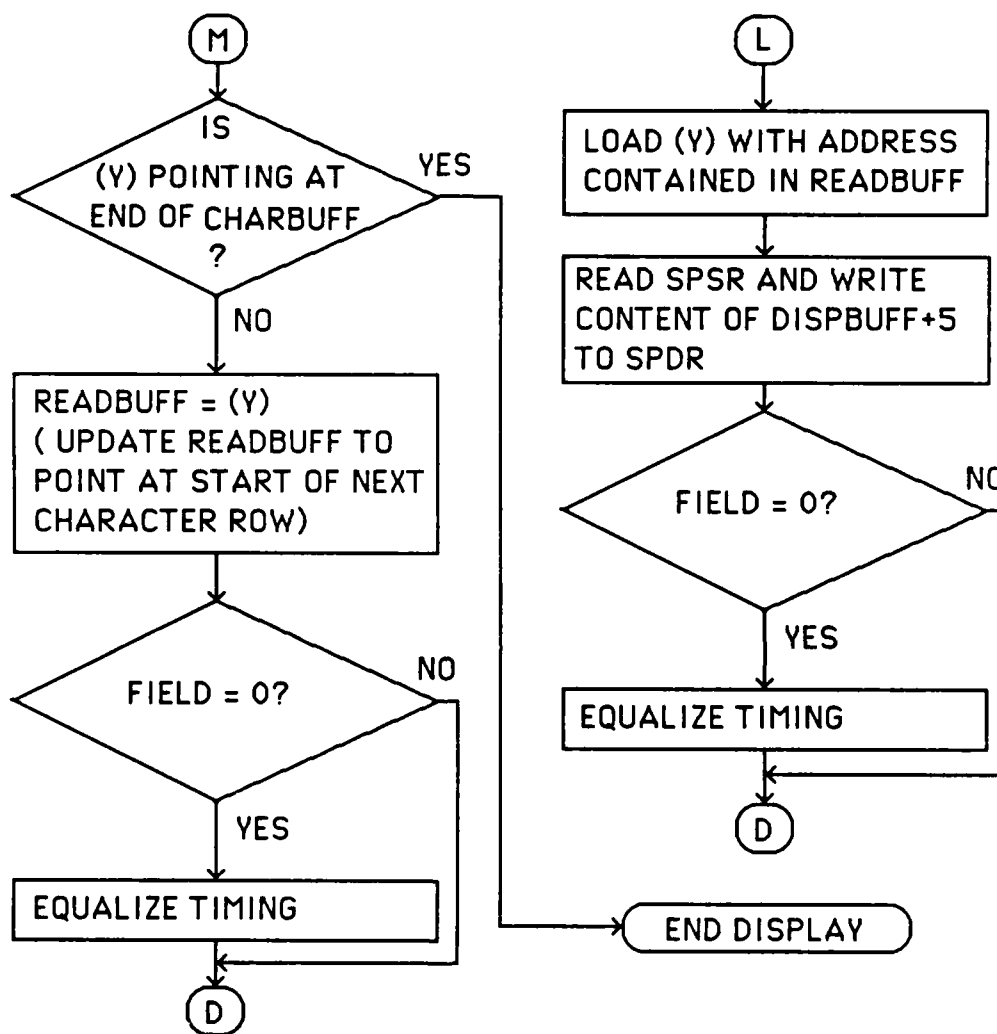


Figure 4-4. (Cont'd)

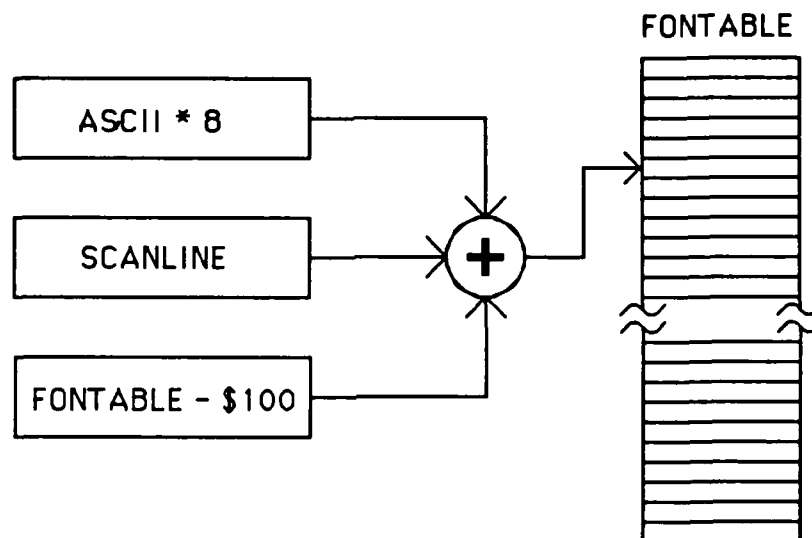


Figure 4 - 5. Computing Font Table Address

Index register X now contains the address of the desired display pattern for the given ASCII code and scanline. Again it is time to send another pattern, this time the content of `DISPBUFF+2`. The value of `SLCNT` is decremented by one and checked to see if it is zero. If it isn't, then index register Y is incremented by one and `FIELD` is checked. Note the same 6 μ s section of code is executed except the condition for executing it is opposite of that for executing it at the start of the loop. This illustrates how the "odd" and "even" frames are shifted with respect to each other by 6 μ s. If `FIELD` does equal "0" here, then index register X points at a display pattern and `DISPTR` points at `DISPBUFF+3`. Thus the first byte in the second half of the display buffer has been filled. The program loops back to generation of the `HSYNC` pulse and incrementing of index register Y. Index register Y has now been

incremented by 2 since the last iteration of the loop. It now points to the next "even" or "odd" address in the character buffer. The same process is repeated until the value of SLCNT reaches "0".

When SLCNT reaches "0", the first three horizontal lines of the first character scanline have been displayed. As the loop is exited, there is still one more pattern to be looked up and placed in DISPBUFF+5 (pointed at by DISPTR). When this accomplished is based on the value of FIELD. Again, this is necessary in order to maintain the proper timing relationships. The fourth and last horizontal line of the scanline is output. At the same time, SLCNT is reinitialized to 3, index register Y is reloaded with the address contained in READBUFF (this points Y back to the start of the same character row in the character buffer). The value of SCANLINE is incremented by one so the proper patterns are looked up. DISPTR is reset to DISPBUFF-1.

The next loop is exactly the same as the first loop with a couple of exceptions. Instead of sending the patterns in DISPBUFF through DISPBUFF+2, the patterns in DISPBUFF+3 through DISPBUFF+5 are sent. Furthermore, because DISPTR starts at DISPBUFF-1 instead of DISPBUFF+2, the new patterns are placed in DISPBUFF through DISPBUFF+2. This loop will also continue until SLCNT reaches "0".

Once again, one more horizontal line must be displayed to complete the scanline. In addition to reinitializing SLCNT to 3, setting DISPTR to DISPBUFF+2, and incrementing SCANLINE, some further checks have to be made. The first additional check that has to be made at this point is to see if all the scanlines for the current character row have been displayed. This is done by checking if SCANLINE

is equal to 9. If it isn't, then index register Y is loaded with address contained in READBUFF, the pattern in DISPBUFF+5 is sent, and program control jumps back to the start of the first loop to display the next scanline. If SCANLINE is equal to 9, then the end of the current character row has been reached. SCANLINE is reset to 1 in anticipation of the start of the next character row. But before the next character row can start, a check must be made to see if all the character rows have been displayed. This is done by checking to see if the value in index register Y is pointing past the end of the character buffer. If it is, then all the rows have been displayed and the active display portion of the frame is complete. If index register Y is not pointing past the end of the character buffer, then it is pointing at the first character of the next row to be displayed. This address is saved in READBUFF for future reference and control passes to the first loop. The entire process repeats until all the rows have been displayed.

The software code is extremely "well tuned". Every instruction is where it is because of the required timing control. Any attempt to modify to code would be extremely tedious and difficult because the timing relationships must be maintained.

4.6 Video/Keyboard Software Integration

This section explains how the video display software is integrated with the keyboard scan and decode software. The modified BUFFALO monitor calls the display software as a subroutine. This subroutine will display the contents of CHARBUFF on the CRT until a key on the keyboard is depressed and decoded. The subroutine will then return the ASCII code of the depressed key in accumulator

A. The BUFFALO monitor will decide what it wants to do with the code. This will be discussed in detail in Chapter 5.

The algorithms for the the keyboard scan/decode and video signal generation have already been presented. Figure 4-6 is a flowchart showing how the two algorithms are integrated together. When BUFFALO calls the subroutine, a VSYNC pulse is output to start the timing of the video signal. The first task accomplished is to check the VALIDKEY variable. If VALIDKEY is true, then the keyboard is checked for no key depression. This prevents rapid repetition of the same key and requires one key be released before another may be depressed. If VALIDKEY is false, or is made false after checking for no key depression, then the keyboard scan and decode routine is executed as previously described. If an implemented key is decoded, then the subroutine is exited. The value of the ASCII code is returned to BUFFALO in accumulator A.

While the keyboard scan and decode is taking place, HSYNC pulses must be emitted at the right time to synchronize with the CRT. The keyscan software uses these HSYNC pulses as load pulses to the 74165. These HSYNC pulses must occur at the appropriate times, so the keyscan software is written around the timing of these pulses. Recall from the explanation of the keyboard scan/decode algorithm that patterns are sent to the keyboard columns until a key depression is detected or all patterns have been sent. This means the number of HSYNC pulses required for the keyboard scan is not known beforehand. However, the number of blank lines required before the start of the active display period is known. This number, referred to as BLKLINEs in the software listing, is loaded into index register Y. Every time an HSYNC pulse is generated during the keyboard scan and decode, this

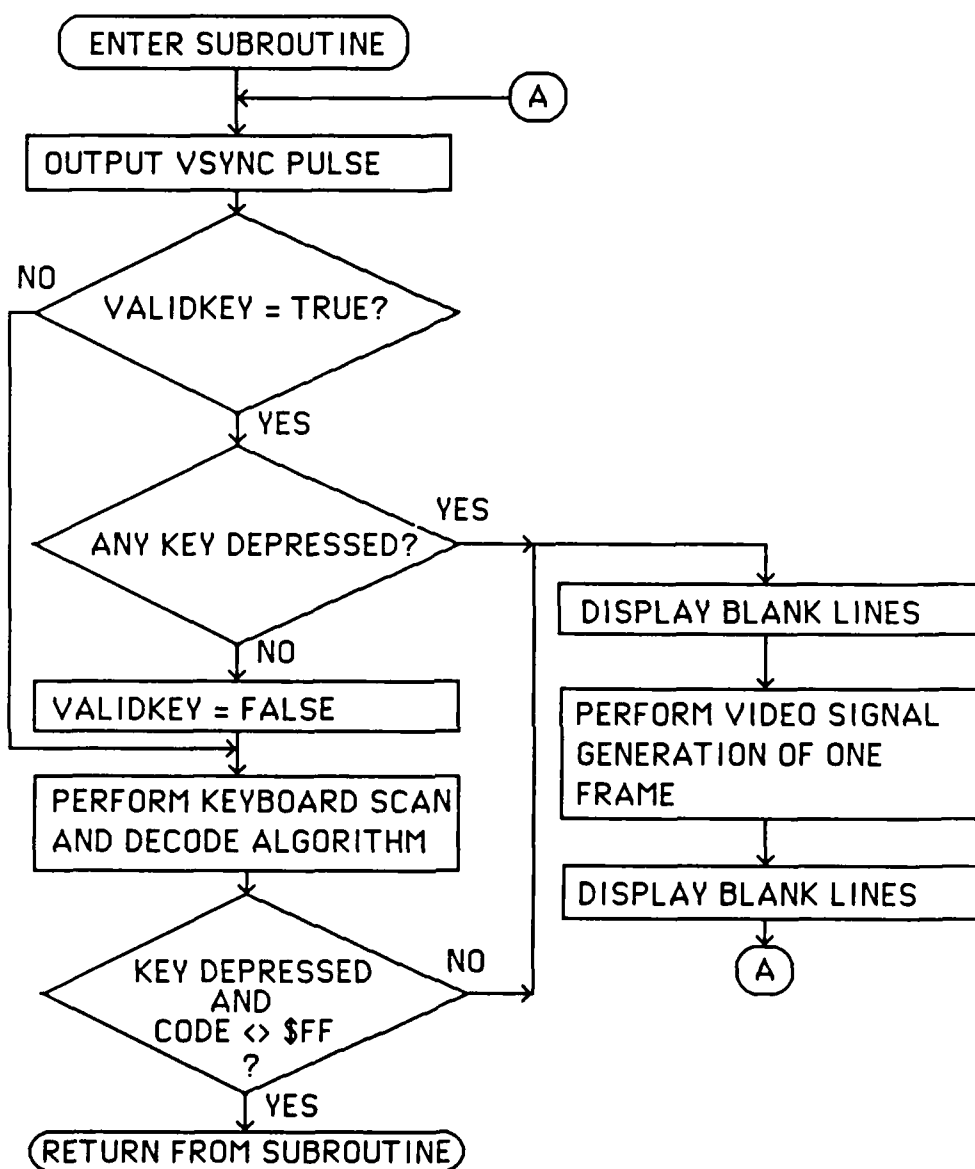


Figure 4 - 6. Keyboard/Video Subroutine Flowchart

value is decremented by one. Thus, upon completion of the scan/decode, index register Y has the number of blank lines that still have to be generated. This number of blank lines is generated prior to beginning the active display period.

The active display period is executed as previously described and more blank lines are displayed to complete the frame. Recall the total number of lines in this design is 254.5 with the active display period requiring 160 horizontal lines. Three line times are used for the VSYNC pulse, 44 lines are used prior to the active display period, and the remainder of the lines are displayed after the active display period. Then the entire process repeats starting with the VSYNC pulse.

4.7 Generating a Composite Video Signal

While some special monitors may have separate inputs for synchronization and video signals, most commercial television sets require an RF modulated composite video signal. In order to get a composite video signal, a simple circuit is used to mix the synchronization signals and the video data. This circuit is shown in Figure 4-7.

The circuit uses two open collector output exclusive NOR gates of a 74266 chip. When the voltage on the MOSI line is low, the current is shunted through the upper 220 Ω resistor resulting in a low voltage on the output line. When the voltage on the MOSI line is high, the diode is forward biased resulting in a voltage increase across the lower 220 Ω resistor. The synchronization pulses arrive on the D5 line. When the D5 line goes low, the output of the gate goes low resulting in a low

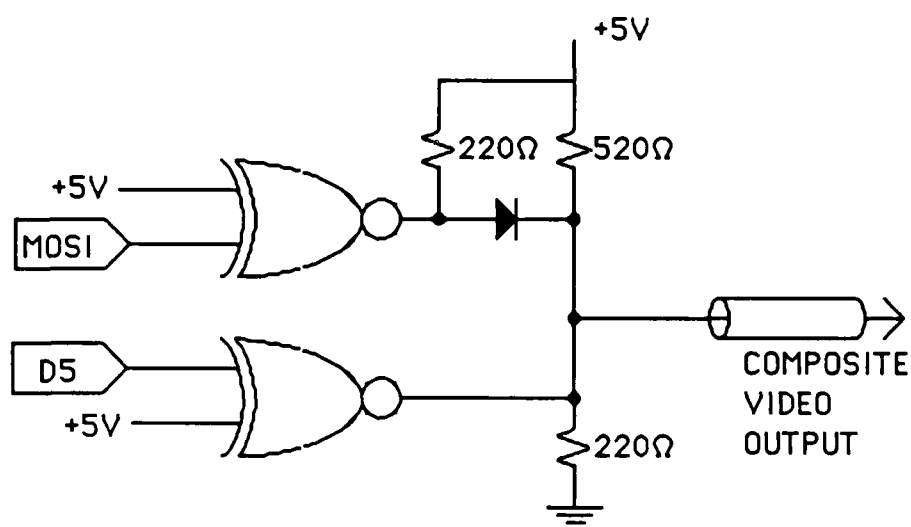


Figure 4 - 7. Composite Video Circuit

voltage at the output of the circuit. Thus, when the D5 line is pulsed low for the HSYNC and VSYNC pulses, the electron beam voltage is essentially at ground (black).

If the television has a direct composite video input, the output of this circuit can be used directly. If there is no direct composite video input, the output of this circuit must be input to an RF modulator and the output of the modulator connected to the television.

Chapter Five

The Modified BUFFALO Monitor

This chapter explains the operation of the modified BUFFALO monitor. First, the determination of the I/O device is explained. This is followed by a presentation of how data in input to and output from the modified BUFFALO monitor. Finally, a summary of the changes to the original BUFFALO monitor is presented. A complete software listing of the modified BUFFALO monitor can be found in Appendix D.

5.1 Determining the I/O Device

Originally, the BUFFALO monitor was designed to accomplish I/O via the onboard SCI, an external ACIA, or a an external DUART. Since this system operates only in the single-chip mode, no external ACIA or DUART may be used. However, the onboard SCI may still be used for I/O. Thus, the option to use the system with a terminal is retained.

The flowchart in Figure 5-1 depicts how BUFFALO determines which device is going to be used for I/O. Both the SPI and SCI are initialized. A variable called IODEV is used to indicate which device will be used. If IODEV is 0, then the SCI is the I/O device. If IODEV is 1, then the SPI is the I/O device. The I/O drivers use IODEV to determine which set of routines to call for I/O processing. Modified BUFFALO will continue to poll both devices until input is received. The first device to respond will be the device used for I/O. The device not used for I/O is available for user software.

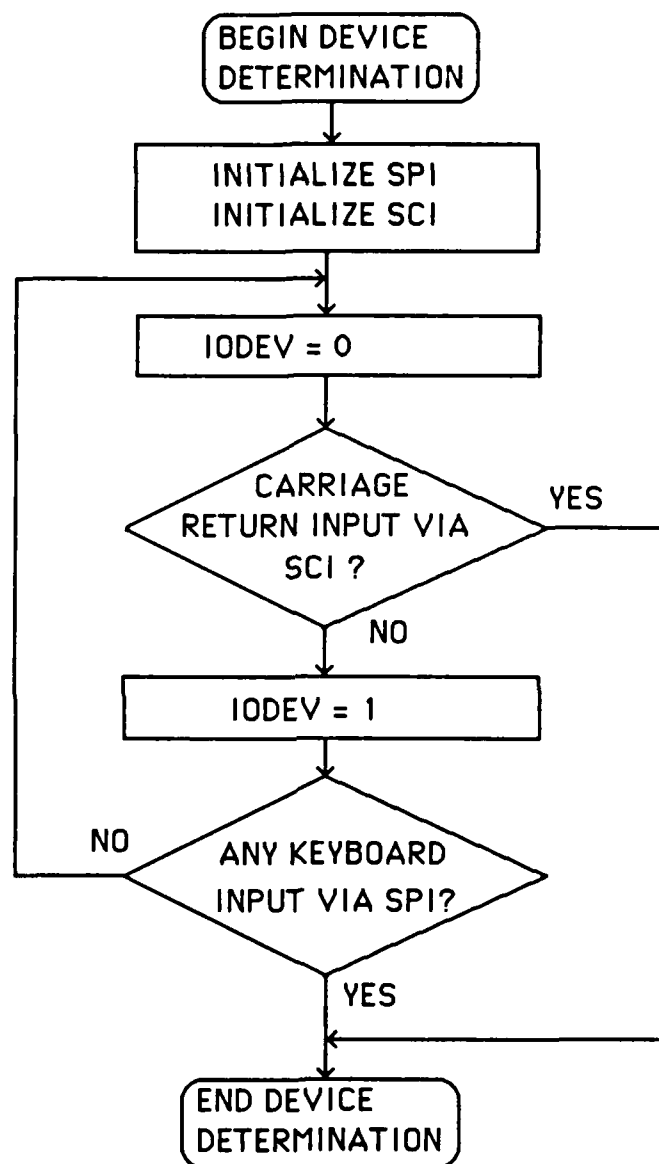


Figure 5-1. Determining the I/O Device

5.2 Data Input to BUFFALO

In order to process commands, BUFFALO must assemble a command line. It assembles this command line by calling the input I/O driver. The input I/O driver checks the value of IODEV to determine which set of routines to use to get a character. If IODEV = 1, then the keyscan/video display subroutine is called and the ASCII character code is returned in accumulator A. If IODEV = 0, then the SCI input routines are called and the ASCII character code is also returned in accumulator A. BUFFALO checks this code to see if it is a backspace, carriage return, or control character. If it isn't, then the character is placed in the character buffer. This is the same 30-byte character buffer as previously described. BUFFALO will continue to call the input driver for more input until a carriage return or control character is encountered. Once a carriage return is entered, BUFFALO assumes a command has been entered. The available commands are explained in Appendix A.

At this point, the character buffer contains the command and any possible arguments. A command table is searched to see if the entered command matches any of the BUFFALO commands. If it does, then BUFFALO calls the command task as a subroutine. The task is accomplished and the user can enter another command. If the command entered does not match any of the commands in the command table, then an error message is displayed and the user can try again.

5.3 Data Output from BUFFALO

The output of data from the modified BUFFALO monitor when using the SCI device is straightforward and unchanged from the original version of

BUFFALO. However, when the device is the SPI, several changes had to be made. This section will discuss those changes.

Many of the BUFFALO commands display some type of data such as contents of memory location or register contents. When using the SPI for I/O, the output data must be put in the 30-byte character buffer for display using the keyscan/video display subroutine. The actual process of putting the characters into the buffer is transparent to the commands. The command software sends its output to the output driver. The output driver calls the output routines which in turn put the data in the character buffer. The amount of the data in the character buffer is kept track of so that if the buffer becomes full, the filling of the character buffer is halted and the user is shown the data in the buffer. This is done by calling the keyscan/video display subroutine when the buffer is full. The user can view the data until a key is depressed on the keyboard. The character buffer is then filled with the rest of the data, if any, and the process continues. The output data is formatted for the 5 line by 6 character display.

5.4 Summary of Changes to the BUFFALO Monitor

This section summarizes the major changes made to the original BUFFALO 2.2 monitor. In addition to the changes discussed in sections 5.1 through 5.3, the following changes were made to the original BUFFALO 2.2 monitor:

- 1) The assembler/disassembler was changed so that EEPROM locations could be assembled/disassembled. In addition, when the assembler/disassembler is expecting input, a "." prompt will be displayed instead of the ">" prompt.

- 2) The Go, Call, and Proceed commands were changed so that in all three cases each instruction is traced instead of inserting SWI's for breakpoints. After each instruction is traced, the program counter is checked to see if it matches one of the breakpoints. This was done to extend the life of the EEPROM because the EEPROM is only good for 400 erasures.
- 3) A command called EX was added so real time applications could be run. The EX command allows the user to run a program without tracing through the program.
- 4) The LOAD command was changed to get data from only the SCI. The LOAD command will only work when the SCI is the I/O device.
- 5) The original BUFFALO 2.2 had dedicated RAM locations for every interrupt vector. This was changed to reduce the amount of RAM used for interrupt vectors. The user supplies a number corresponding to a particular interrupt and the address of the interrupt handler. There is a routine in ROM which is entered when an interrupt is generated. This routine checks the user vector number against the type of interrupt which occurred to determine if the user has an interrupt handler for the particular interrupt. If it does, then control passes to the interrupt handler. If it doesn't, then an RTI is executed.

Chapter Six

Summary

This thesis has presented a low cost video display system using the M6811 single chip microcomputer. The design of the software algorithms and the hardware necessary to implement the system have been presented and discussed. The system has been built and meets its desired objective of providing a low cost display system that students can use to learn about the M6811.

In the course of the design, several problems had to be overcome. One major problem was the limited speed of the SPI. The design of an "interlaced" display with half of the characters displayed during each alternating frame allowed an increased effective bandwidth at the expense of a tolerable flicker of the display. In addition, because of the limit of 8K of onboard ROM, the display software had to be optimized to use repetitive loops whenever possible to reduce overall program size. This allowed the display software along with the BUFFALO monitor to fit within the limits of the 8K of onboard ROM. Because of the strict timing requirements of the video signals, the software had to be well "tuned". Every clock cycle had to be counted to ensure the right output was generated at the right time. While not extremely difficult in nature, this process was very tedious.

Finally, this thesis has shown the viability of using the M6811 to read a keyboard and generate the necessary signals to produce a composite video signal. Hopefully, the system will provide an excellent learning tool for undergraduate students studying microprocessors.

Appendix A

A User's Guide to the Modified BUFFALO Monitor

This appendix contains information necessary to user the modified BUFFALO monitor with the low cost video display system.

A.1 System Startup

When the system is powered up, the television screen will consist of a series of lines across the screen. Press the RETURN key on the keyboard and the prompt "BUFFALO 2.2" will be displayed. Now press any key and the BUFFALO prompt , ">", will be displayed. Commands may be entered any time the ">" prompt is displayed.

A.2 Available Memory for Programs

Care must be taken when entering data into the memory. If certain locations are altered, the monitor will crash.

The following memory locations are available:

RAM	00A1 - 00FF	95 bytes
EEPROM	B648 - B7FF	440 bytes

Thus, there are 535 bytes available for program development. Be sure to use the above memory locations.

A.3 Interrupt Routines

The M6811 has a vector for each possible interrupt located at upper addresses of the 8K ROM. Since the vectors for almost all of these interrupts are not known beforehand, a method had to be devised to allow the user to specify the address of the interrupt handler.

There are two vectors available in RAM for the user interrupt handlers. The modified BUFFALO monitor checks these two vectors to determine if the user has interrupt handler for the interrupt. The user supplies the vector number and address in the designated RAM locations. These are the designated RAM locations for vector number and address:

Vector Number 1 - Vector Number: 0095 Address: 0096

Vector Number 2 - Vector Number: 0098 Address: 0099

The Vector Numbers and their associated interrupts are as follows:

<u>Vector Number</u>	<u>Interrupt</u>
1	SCI
2	SPI
3	Pulse Accumulator Input Edge
4	Pulse Accumulator Overflow
5	Timer Overflow
6	Timer Output Compare 4
7	Timer Output Compare 3

8	Timer Output Compare 2
9	Timer Output Compare 1
10	Timer Input Compare 3
11	Timer Input Compare 2
12	Timer Input Compare 1
13	Real Time Interrupt
14	IRQ
15	XIRQ
16	Illegal Opcode Trap
17	COP Failure
18	COP Clock Monitor Fail

As an example, suppose a program has an interrupt handler routine for the Timer Output Compare 3 located at address B70E. The number 7 would be loaded into either of the two Vector Number locations and the address B70E would be loaded into the associated address location. If Vector Number 1 were used, then the vector memory locations would look like:

0095 - 07

0096 - B7

0097 - 0E

When an interrupt from the Timer Output Compare 3 is generated, the interrupt handler at B70E will be executed.

In addition , there are two dedicated interrupts for the Timer Output Compare 5 and the Software Interrupt. Each of these requires 3 bytes and are located as follows:

Timer Output Compare 5: 009B

Software Interrupt: 009E

In order to use these, the JMP opcode (\$7E) is placed in the first byte and the address of the interrupt handler is placed in the last two bytes.

A.4 Modified BUFFALO Commands

In keeping with Motorola documentation, the command line format is as follows:

><command>[<parameters>] [RETURN]

where:

<u>SYMBOL</u>	<u>FUNCTION</u>
>	BUFFALO monitor prompt
<command>	Command mnemonic
<parameters>	Expression or address
[<parameters>]	Optional parameters
[RETURN]	RETURN keyboard key depressed

General Information:

1. Fields are separated by at least one space.

2. All numbers are interpreted as hexadecimal. Do not use a \$ to precede hexadecimal numbers.
3. A maximum of 28 characters may be entered on a command line. If more characters are entered then the error message "Too Long" will be displayed.
4. The BREAK key will abort a command.

The following is a list of the commands, their function, and page number:

<u>COMMAND</u>	<u>FUNCTION</u>	<u>PAGE</u>
ASM	Assembler/Disassembler	62
BR	Set breakpoint	63
BR-	Remove breakpoint	63
CALL	Call subroutine	64
EX	Execute program	64
G	Go execute program	64
HELP	Display commands	65
LOAD	Load program	65
MD	Memory display	66
MM	Memory modify	66
MV	Move memory block	67
P	Proceed from breakpoint	67
RD	Register Display	68
RM	Register Modify	68
T	Trace	68

ASM ASSEMBLER/DISASSEMBLER

This command allows the user to assemble and disassemble code one line at a time. The format for the command is as follows:

ASM <address> [RETURN]

The <address> is the address of the first byte of the line to be disassembled. After the RETURN key has been depressed, the mnemonic along with the arguments (if any) will be displayed on the screen. The disassembled line will be displayed on the screen until the user depresses any key. The screen will then be clear except for a "." prompt displayed in the upper left corner of the display. The "." prompt indicates BUFFALO is waiting for an input line to assemble. If no input is needed, type [RETURN] and the next line will be disassembled and displayed. To terminate the ASM, depress the BREAK key.

Notes:

- a '#' preceding an argument indicates immediate addressing
- arguments must be separated by a space
- a ',' indicates indexed addressing and the next character must be X or Y

Subcommands (entered at the end of the command line):

[RETURN] - Disassembles the next opcode. If there was no assembly input, the next opcode is retrieved from the disassembler.

[LINE FEED] - Same as [RETURN] except if there is no assembly input, the <address> is incremented and this new address is disassembled.

[UP ARROW] - same as [LINE FEED] except <address> is decremented.

[BACK SLASH] (/) - reassembles current <address>.

BR Breakpoint Set

This command allows breakpoints to be put into the breakpoint table. The following is the command format for this command:

BR [<address>] [RETURN]

If <address> is provided, it is placed in the breakpoint table if the table is not full and the contents of the updated breakpoint table are displayed. If all four entries in the breakpoint table are used, then the message "FULL" will be displayed and the breakpoint will not be entered in the table. If <address> is not provided then the contents of the breakpoint table will be displayed. More than one address may be provided on the command line, but once the table has four entries, no further addresses will be entered into the table and the "FULL" message will be displayed.

BR - Remove Breakpoint

This command removes breakpoints from the breakpoint table. The following is the command line format:

BR -[<address>] [RETURN]

If <address> is provided, and it is in the breakpoint table, the address is removed from the table. Just as with BR, multiple addresses may be entered on a single command line (each address must be preceded by a "-"). If no <address> is provided, then all the breakpoints in the table will be removed.

CALL Call Subroutine

This command allows the user to call a subroutine. Control will return to the monitor when either a RTS or breakpoint is encountered. The command line format is as follows:

CALL [<address>] [RETURN]

If <address> is not supplied, then the user's PC value is used as the starting address. Like the Go command, Call will not work properly with a real time application such as timing external events.

EX Execute Program

This command allows the user to execute a program without breakpoints or the limitations of the GO command. This command should be used when attempting to run a real time application or an application which must get precise event timing. The GO command will ruin any attempt to accurately time events. The command line format is as follows:

EX [<address>] [RETURN]

If <address> is not supplied, then the user's PC value is used as the starting address.

G Go execute program

This command executes a user's program. It will return control to BUFFALO if a breakpoint is encountered. This command works by invoking the TRACE command until a breakpoint is encountered. Thus, any program that relies on timing events will not work properly. Therefore, use the EX command when

precise timing is required. The command line format for this command is as follows:

G [<address>] [RETURN]

If <address> is not provided, then execution begins at the current value of the Program Counter.

HELP Display commands

This command displays the commands available in the BUFFALO monitor.

The command line is as follows:

HELP [RETURN]

LOAD Load program

Load allows a user program to be downloaded to the 6811. This command only works when using a computer with a communications software application such as a Macintosh with MacTerminal. This command will not work with the CRT display. The following is the command line format:

LOAD [RETURN]

After the command is entered, BUFFALO waits for an S-record format file to be sent (for more information on S-record format see [MOT85B]). In MacTerminal, the SEND FILE command is used to send the file to BUFFALO. The baud rate of the terminal must be set at 600 for proper operation. This is because the data may occupy the EEPROM and it takes about 10 ms to program one byte in the EEPROM.

MD Memory Display

This command permits display of a block of memory. The command line format is as follows:

MD [<address1>][<address2>] [RETURN]

This command will display the contents of memory starting at <address1> up to and including <address2>. If <address2> is not specified, then 128 bytes starting at <address1> will be displayed. If neither <address1> or <address2> are specified, then 128 bytes starting at the default memory pointer location will be displayed. Depressing the BREAK key will abort the MD command.

MM Memory Modify

This command allows the contents of a memory location to be modified. The command line format is as follows:

MM [<address>] [RETURN]

If <address> is not supplied, then the default memory pointer location will be opened for display and modification. If <address> is supplied, then the contents of <address> will be opened for display and modification. If data is entered, the contents of the location are updated to reflect the new data. There are several subcommands associated with the command. These subcommands are entered after the data, if any, is entered.

[RETURN] - Closes the current location and opens the next address.

[-] (minus sign) - Closes the current location and opens the previous address.

[=] (equal sign) - Reopens the current location.

[.] (period) - Closes the current location and exits the memory modify command.

If an attempt is made to modify a ROM location, then BUFFALO will display the message "ROM-". A shortcut to enter the MM mode is to enter "/" when the BUFFALO prompt is displayed. This will open the default memory pointer location. As always, depressing BREAK will abort the command.

MV Move Memory Block

This command allows a block of memory to be moved from one location to another. The command line format is as follows:

MV <source1> <source2> [<destination>] [RETURN]

This would move a block of memory starting at <source1> and ending at <source2> to <destination>. Both <source1> and <source2> must be supplied or an error message will be displayed. If <destination> is not supplied, then the block will be moved up one byte. When using the CRT display system, the screen may go blank because of the length of time required to move blocks to the EEPROM.

P Proceed from breakpoint

This command allows the user to proceed from a breakpoint. It is used after Go or Trace to continue execution of a program. The command line format is as follows:

P [RETURN]

RD Register Display

This command displays the contents of user's CPU registers. The command line format is as follows:

RD [RETURN]

RM Register Modify

This command displays the contents of the CPU registers and allows the user to modify the contents. The command line is as follows:

RM [<register name>] [RETURN]

The allowable register names are P, Y, X, A, B, C, S. If no <register name> is specified, then the first register is opened - the program counter P. The user may then change the value of the register. There are two subcommands:

[RETURN] - Opens the next register.

[.] (period) - Exits the register modify command.

The value of a register is changed only if data is entered before a subcommand.

T Trace

This command allows the user to trace through a program. The command line format is as follows:

T [<count>] [RETURN]

If <count> is not specified, only one instruction will be traced. If <count> is specified, then the trace command will be invoked <count> times. After each instruction has been traced, the contents of the registers are displayed. The BREAK key may be used to abort the command.

A.5 Debugging with the Modified BUFFALO Monitor

Debugging with the modified BUFFALO monitor is very much the same as debugging with any other monitor. The user can set breakpoints and Go or Trace through the program. However, because of the limited number of times the EEPROM may be erased (about 400), SWI instructions are not used to set breakpoints. This way SWI's are not written and erased every time a Go, Proceed, or Call command is used. Instead, each instruction is traced and the Program Counter is checked to see if a breakpoint is hit. This will present no problem to the user unless a real time application is being designed. In this case, such as counting clock cycles between events, the Trace, Go, Call, and Proceed commands will give invalid results to the user. To run a program in real time use the EX command.

Appendix B

Hardware Schematics

This appendix contains the hardware schematic for the low cost video display system.

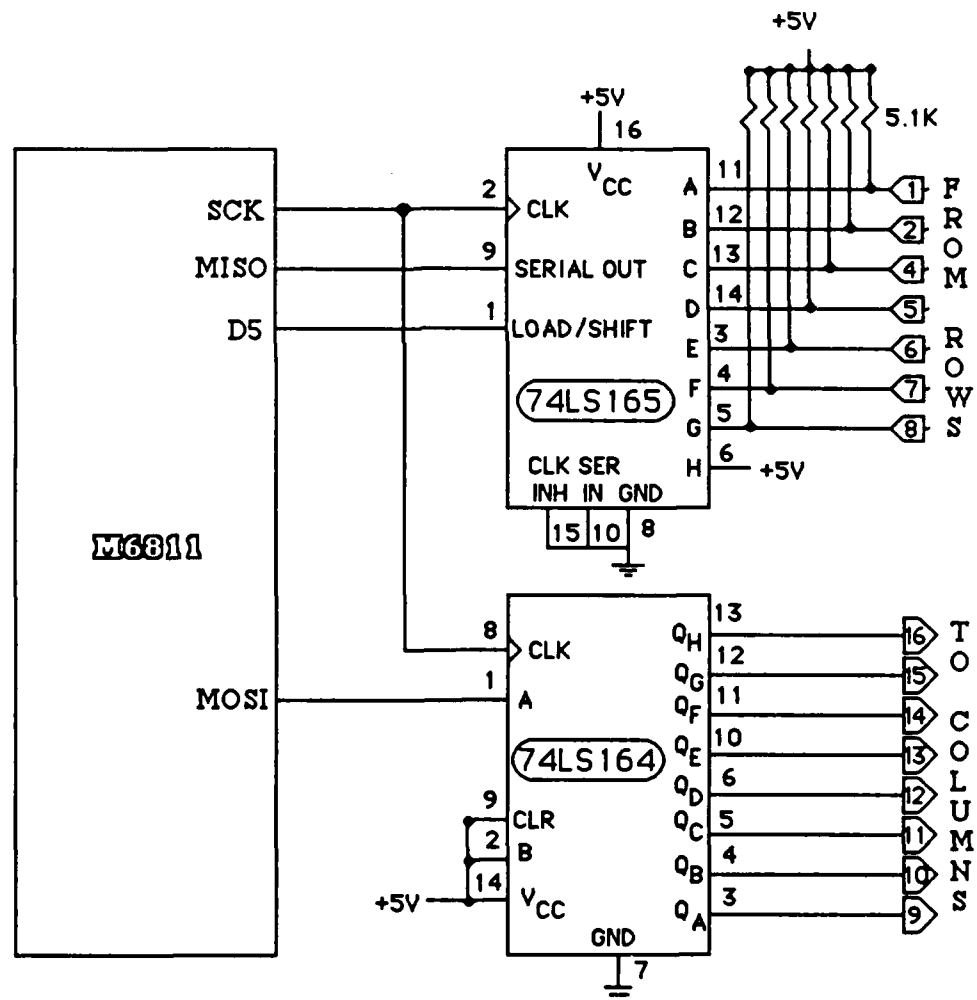


Figure B-1 Keyboard Interface

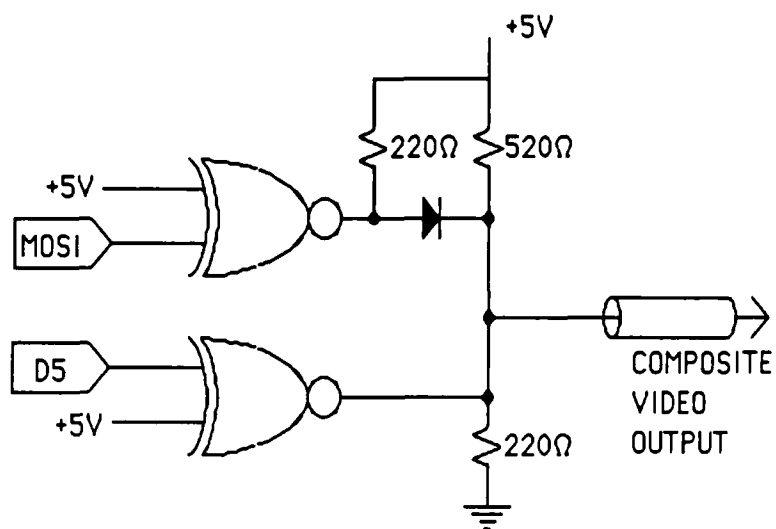


Figure B - 2. Composite Video Circuit

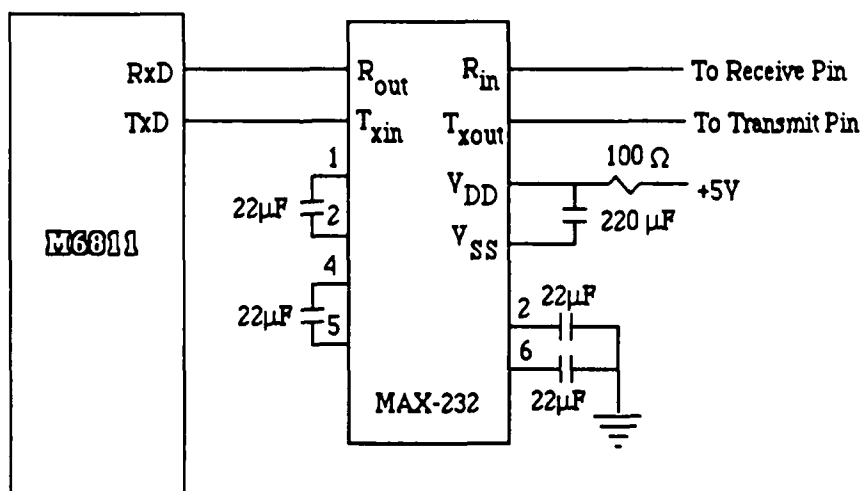


Figure B - 3. RS-232 Circuit

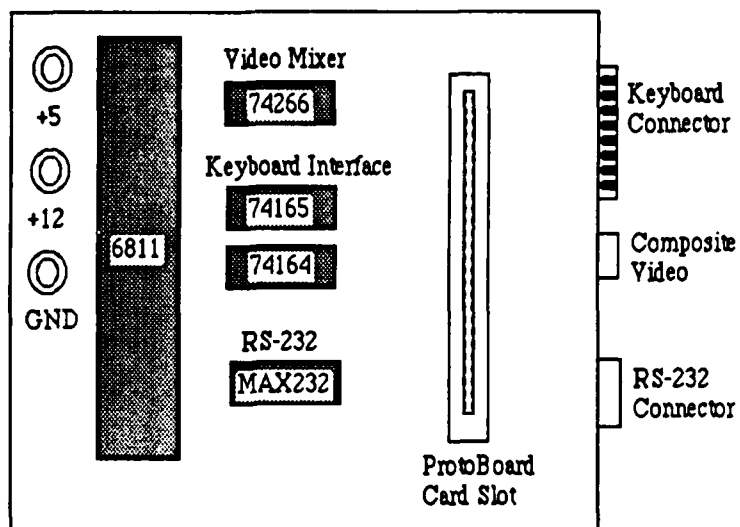


Figure B - 4. Possible Experiment Board Layout

Appendix C

Using a Different Keyboard

In order to provide the system with more flexibility, the capability to use a different keyboard is provided. There are two requirements for a replacement, non-encoded keyboard: it can have a matrix no larger than 8 X 8 and the shift key must result in the largest value for $8 * \text{ROWCNT} + \text{COLCNT}$ (i.e. the shift key must be connected to the most significant bit on the column and the most significant bit on the row). This section describes the changes to the software to use a different keyboard.

The keyboard lookup table is located in the first part of the EEPROM memory area starting at address \$B600. The first six locations are used to define the characteristics of the keyboard. These values are needed by the decoding routines. These locations and names of these six values are as follows:

<u>Address</u>	<u>Name</u>	<u>Purpose</u>
\$B600	SHIFTKEY	Defines the matrix value of the shift key
\$B601	SHIFTLOW	Defines the matrix value of first shiftable key
\$B602	SHIFTHI	Defines the matrix value of the last shiftable key
\$B603	SHFTCOLP	Defines the column pattern for the shift key
\$B604	SHFTROWP	Defines the row pattern for the shift key
\$B605	SHFTRNGE	Defines the offset added for a shifted key
\$B606	TABLE	Start of keyboard lookup table

The best way to illustrate these how these values are determined is by showing how the values for the present keyboard are determined.

As stated earlier, the keyboard used in the system is from a Radio Shack Color Computer . The schematic for the keyboard is shown in Figure C-1. The keyboard consists of a 7 row by 8 column matrix. Note the shift key is in the lower right corner of the matrix. It is connected to the most significant bit of the column pattern and its row line is, in essence, connected to the most significant bit of the row pattern. Each key position has a value associated with it. Starting in the upper left corner, the key has a matrix value of 0. The key to its right has a value of 1, and so on, until the end of the first row is reached with a value of 7. The next key after 7 is the start of the second row. Thus, for this keyboard the SHIFTKEY has a matrix value of 55.

The next characteristics are SHIFLOW, SHIFTHI, and SHFTRNGE. There are no lower case letters in the display font, so there is no need to use the shift key with the alphabetic characters. This means the first shiftable key is the 1 key. Its matrix value is 33 ($4 \times 8 + 1$). Thus, SHIFLOW has a value of 33. Similarly, the last shiftable key value, SHIFTHI, is 47. To find SHFTRNGE, notice the last key (exclude the shift key) has matrix value 50. The shifted ASCII codes must begin in the table after the value for matrix key 50. To point at the shifted value, an offset of 18 ($51 - 33$) must be added to the unshifted value. Therefore, SHFTRNGE is set at 18. This scheme results in minimum memory used for the lookup table.

The final two characteristics are SHFTCOLP and SHFTROWP. These values are necessary to check for shift key depression when another key has been depressed. The value for SHFTCOLP consists of a zero in the column of the shift

key and ones in the rest of the bits. For this keyboard, the value of SHFTCOLP is \$7F (0111 1111). The value for SHFTROWP is the complement of a zero in the row of the shift key with ones in the other bits. For this keyboard, the value is the complement of \$BF (1011 1111). This value is \$40 (0100 0000).

The table of ASCII codes starts at address \$B606. This address is referred to as TABLE in the software. The table starts with the ASCII code for @ and continues with A, B, C, D, E, F, G, H, I, and so on until the BRK code. Immediately following the BRK ASCII code are the ASCII codes for the shiftable keys, starting with ! and ending with ?.

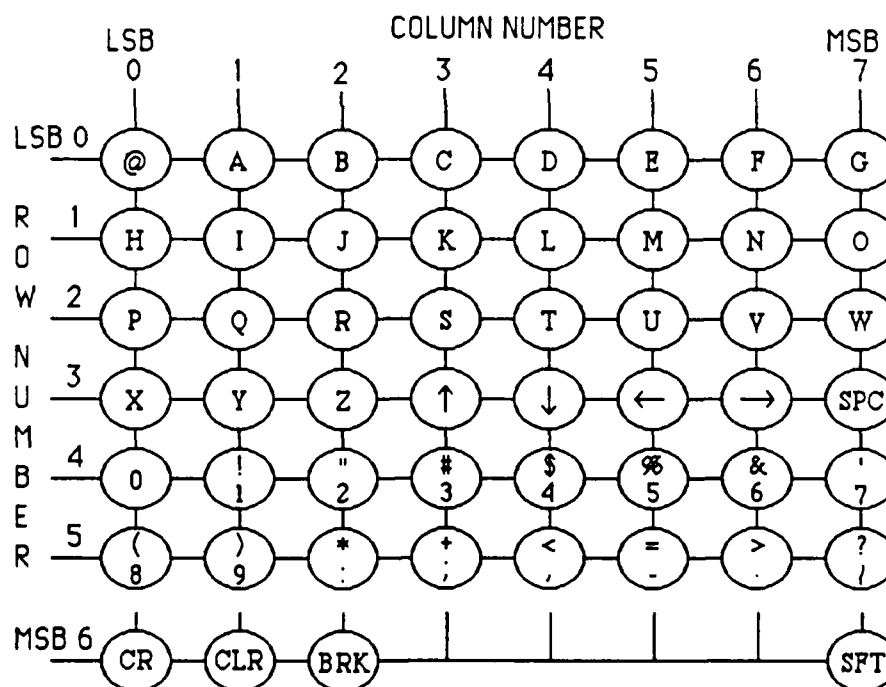


Figure C - 1. Radio Shack Keyboard Schematic

One final note about using a different keyboard. Be sure any unused inputs to the 74LS165 are tied high, not low. This is a must for the keyboard decoding software to work properly. For example, if the new keyboard only has six row lines then the G and H inputs to the 74LS165 must be tied high.

Appendix D

Software Listing

This appendix contains the software listing. The entire modified BUFFALO 2.2 monitor is included for reference. The keyscan/video display software is under the name VIDEO and starts at line 3499. The portions of the original BUFFALO 2.2 monitor are reprinted courtesy of Motorola, Inc. The following routines were written to support the video display system:

<u>Name</u>	<u>Line Number</u>
INITLP	0120
CRTBKSPC	0318
INITCRT	0353
OUTCRT	0415
CRTSTRG	0491
CLRBUFF	0507
HORPULSE	0525
EX	1512
VIDEO	3499
FONTABLE	3960
VECTCHK	4238

```

0001 *****
0002 ***** SYMBOLIC DEFINITIONS *****
0003 *****
0004
0005 0000 RAMBS EQU $0 start of RAM
0006 1000 REGBS EQU $1000 start of register set
0007 E000 PGMSTRT EQU $E000 start of ROM
0008 1008 PORTD EQU REGBS+$08 port D I/O register
0009 1009 DDRD EQU REGBS+$09 data dir reg for port D
0010 100A PORTE EQU REGBS+$0A port e
0011 100E TCNT EQU REGBS+$0E timer count
0012 101E TOC5 EQU REGBS+$1E oc5 reg
0013 1020 TCTL1 EQU REGBS+$20 timer control 1
0014 1022 TMSK1 EQU REGBS+$22 timer mask 1
0015 1023 TFLG1 EQU REGBS+$23 timer flag 1
0016 1024 TMSK2 EQU REGBS+$24 timer mask 2
0017 1028 SPCR EQU REGBS+$28 SPI Control Register
0018 1029 SPSR EQU REGBS+$29 SPI Status Register
0019 102A SPDR EQU REGBS+$2A SPI Data Register
0020 102B BAUD EQU REGBS+$2B SCI baud reg
0021 102C SCCR1 EQU REGBS+$2C SCI control1 reg
0022 102D SCCR2 EQU REGBS+$2D SCI control2 reg
0023 102E SCSR EQU REGBS+$2E SCI status reg
0024 102F SCDAT EQU REGBS+$2F SCI data reg
0025 1039 OPTION EQU REGBS+$39 option reg
0026 103A COPRST EQU REGBS+$3A
0027 103B PPROG EQU REGBS+$3B ee prog reg
0028 103C HPRI0 EQU REGBS+$3C hprio reg
0029 103F CONFIG EQU REGBS+$3F config register
0030 001E BUFFLEN EQU 30 buffer holds 30 chars
0031 0004 EOT EQU $04 end of transmission
0032 0008 BACKSPACE EQU $08 ASCII backspace
0033 0018 CTLX EQU $18 control x
0034 0002 CTLA EQU $02
0035 0017 CTLW EQU $17
0036 007F DEL EQU $7F abort
0037 003F SWI EQU $3F
0038 000D CRETURN EQU $0D ASCII carriage return
0039 0018 KEYBREAK EQU CTLX
0040 0020 SPACE EQU $20 ASCII space
0041 003E PROMPT EQU '>'
0042 0020 SSHIGH EQU $20
0043 00DF SSLOW EQU $DF
0044 003A DDRDINIT EQU $3A
0045 005C CTRLINIT EQU $5C
0046 002C BLKLINES EQU 44 # of blank lines which
0047 * * comprise the V Front porch
0048 005A TOTBLKS EQU 90 # blank lines per frame
0049 0003 SLINES EQU 3
0050 0008 MAXROWS EQU 8 # of keyboard rows
0051 *-----

```

```

0052      *      RAM STORAGE DEFINITIONS
0053      *-----*
0054 0000      ORG  $0      start at RAM location 0
0055 0000      VALIDKEY  RMB  1      1 = good key decoded
0056 0001      IODEV    RMB  1      0=SCI : 1=SPI
0057 0002      CHARPTR   RMB  1      CHARBUFF pointer
0058 0003      READBUFF  RMB  2
0059 0005      CHARBUFF  RMB  BUFFLEN  char buff holds 30 chars
0060 0023      ENDBUFF   EQU  *
0061 0023      DISPTR    RMB  2      DISPBUF pointer
0062 0025      DISPBUFF  RMB  6      double line display buffer
0063 002B      FIELD     RMB  1      0=even frame : 1 = odd
0064 002C      SLCNT     RMB  1      # of H lines per scan line
0065 002D      RMB  20      user stack area
0066 0041      USTACK    RMB  30      monitor stack area
0067 005F      STACK     RMB  1
0068 0060      COMBUFF   RMB  8      command buffer
0069 0068      SHFTREG    RMB  2
0070 006A      BRKTABL   RMB  8
0071 0072      REGS      RMB  9
0072 007B      SP        RMB  2
0073 007D      AUTOLF    RMB  1
0074 007E      COUNT     RMB  1
0075 007F      PTRMEM    RMB  2
0076 0081      PTR0      RMB  2
0077 0083      PTR1      RMB  2
0078 0085      PTR2      RMB  2
0079 0087      PTR3      RMB  2
0080 0089      PTR4      RMB  2
0081 008B      PTR5      RMB  2
0082 008D      PTR6      RMB  2
0083 008F      PTR7      RMB  2
0084 0091      TMP1      RMB  1
0085 0092      TMP2      RMB  1
0086 0093      TMP3      RMB  1
0087 0094      TMP4      RMB  1
0088
0089      *++++++ INTERRUPT VECTORS ++++++
0090 0095      VECTR1     RMB  1      user vector number 1
0091 0096      VJMP1      RMB  2      user vector number 1 address
0092 0098      VECTR2     RMB  1      user vector number 2
0093 0099      VJMP2      RMB  2      user vector number 2 address
0094 009B      VOC5       RMB  3      vector for OC5 interrupt
0095 009E      VSWIIN     RMB  3      vector for SWIIN
0096 0101      USERRAM    EQU  *      start of user RAM
0097
0098
0099
0100
0101      *****
0102      *      INITIALIZATION      *

```



```

0103          *****
0104          E000          ORG   PGMSTRT
0105          E000  86 93   LDAA  #$93
0106          E002  B7 10 39 STAA  OPTION
0107          E005  4F      CLRA
0108          E006  B7 10 24 STAA  TMSK2
0109          E009  8E 00 5F LDS   #STACK      monitor sp
0110          E00C  CE 00 41 LDX   #USTACK
0111          E00F  DF 7B   STX    SP          default user sp
0112          E011  86 D0   LDAA  #$D0
0113          E013  97 7A   STAA  REGS+8      default ccr
0114          E015  BD E3 6C JSR   BPCLR      clear breakpoints
0115          E018  7F 00 7D CLR   AUTOLF
0116          E01B  7C 00 7D INC   AUTOLF      auto cr/lf = on
0117          * initialize devices
0118          E01E  BD E1 85 JSR   INTCRT      initialize crt system
0119          E021  BD E3 CC JSR   ONSCI      initialize SCI system
0120          E024  7F 00 01 INITLP CLR   IODEV      (0 = SCI)
0121          E027  BD E1 41 JSR   INPUT      read SCI
0122          E02A  81 0D   CMPA  #CRETURN      check for carriage return
0123          E02C  27 0A   BEQ   SIGNON      found device - the SCI
0124          E02E  7C 00 01 INC   IODEV      (1 = SPI)
0125          E031  BD E1 98 JSR   CRTCHK      check for a key
0126          E034  81 00   CMPA  #0
0127          E036  27 EC   BEQ   INITLP      no input from SPI;try again
0128          E038  CE E4 85 SIGNON LDX   #MSG1
0129          E03B  BD E2 08 JSR   OUTSTRG
0130
0131          *+++++*****
0132          *++++ Main Loop - Gets input from user  +++++
0133          *++++ Assembles a command line  +++++
0134          *++++*****
0135
0136          E03E  8E 00 5F MAIN   LDS   #STACK      initialize monitor stack
0137          E041  BD E1 F7 JSR   OUTCRLF
0138          E044  BD E2 4E JSR   CLRBUFF      clear buffer for use by CRT
0139          E047  86 3E   LDAA  #PROMPT      prompt user
0140          E049  BD E1 76 JSR   OUTPUT      send it to device
0141          E04C  BD E1 37 MAIN10 JSR   INCHAR      get character from user
0142          E04F  CE 00 05 LDX   #CHARBUFF
0143          E052  D6 02   LDAB  CHARPTR
0144          E054  3A      ABX          offset into buffer
0145          E055  81 18   CMPA  #CTLX      abort?
0146          E057  27 E5   BEQ   MAIN      yes - restart
0147          E059  81 08   CMPA  #BACKSPACE backspace?
0148          E05B  26 10   BNE  MAIN20
0149          E05D  96 01   LDAA  IODEV      Is I/O device SPI or SCI?
0150          E05F  27 05   BEQ   MAIN11
0151          E061  BD E1 64 JSR   CRTBKSPC
0152          E064  20 E6   BRA   MAIN10
0153          E066  7A 00 02 MAIN11 DEC   CHARPTR

```

0154	E069	2D D3		BLT	MAIN	
0155	E06B	20 DF		BRA	MAIN10	
0156	E06D	81 0D	MAIN20	CMPA	#CRETURN	carriage return?
0157	E06F	26 14		BNE	MAIN30	no
0158	E071	7D 00 01		TST	IODEV	
0159	E074	26 07		BNE	MAIN21	
0160	E076	5D		TSTB		
0161	E077	27 C5		BEQ	MAIN	
0162	E079	A7 00		STAA	0,X	
0163	E07B	20 22		BRA	COMM0	
0164	E07D	C1 02	MAIN21	CMPB	#2	any characters in the buffer?
0165	E07F	27 BD		BEQ	MAIN	no - start over
0166	E081	A7 00		STAA	0,X	put cr into buffer
0167	E083	20 1A		BRA	COMM0	parse input
0168	E085	7D 00 01	MAIN30	TST	IODEV	
0169	E088	26 05		BNE	MAIN35	
0170	E08A	A7 00		STAA	0,X	
0171	E08C	5C		INCB		
0172	E08D	D7 02		STAB	CHARPTR	
0173	E08F	C1 1E	MAIN35	CMPB	#BUFFLEN	
0174	E091	2D 08		BLT	MAIN40	
0175	E093	CE E4 97		LDX	#MSG3	
0176	E096	BD E2 08		JSR	OUTSTRG	
0177	E099	20 A3		BRA	MAIN	
0178	E09B	81 2D	MAIN40	CMPA	#/'	
0179	E09D	26 AD		BNE	MAIN10	
0180						
0181				*+++++		
0182				*++++Parse out and evaluate the command line++++		
0183				*+++++		
0184						
0185	E09F	4F	COMM0	CLRA		
0186	E0A0	97 91		STAA	TMP1	
0187	E0A2	97 68		STAA	SHFTREG	
0188	E0A4	97 69		STAA	SHFTREG+1	
0189	E0A6	5F		CLRB		
0190	E0A7	CE 00 05		LDX	#CHARBUFF	point (X) at input buffer
0191	E0AA	96 01		LDAA	IODEV	
0192	E0AC	27 01		BEQ	COMM01	
0193	E0AE	08		INX		point (X) past prompt char
0194	E0AF	DF 81	COMM01	STX	PTR0	save pointer
0195	E0B1	BD E3 8D		JSR	WSKIP	find first character
0196	E0B4	BD E3 78	COMM1	JSR	RDBUFF	read from buffer
0197	E0B7	CE 00 60		LDX	#COMBUFF	
0198	E0BA	3A		ABX		
0199	E0BB	BD E2 74		JSR	UPCASE	convert to upper case
0200	E0BE	A7 00		STAA	0,X	put in command buffer
0201	E0C0	81 0D		CMPA	#\$0D	
0202	E0C2	27 35		BEQ	SRCH	jump if cr
0203	E0C4	BD E3 9B		JSR	WCHK	
0204	E0C7	27 30		BEQ	SRCH	jump if wspace

0205	E0C9	BD E3 7F		JSR	INCBUFF	move buffer pointer
0206	E0CC	5C		INCB		
0207	E0CD	C1 08		CMPB	#\$8	
0208	E0CF	2F 09		BLE	COMM2	
0209	E0D1	CE E4 97		LDX	#MSG3	"long"
0210	E0D4	BD E2 08		JSR	OUTSTRG	
0211	E0D7	7E E0 3E		JMP	MAIN	
0212	E0DA	81 2F	COMM2	CMPA	#/'	
0213	E0DC	26 16		BNE	COMM4	jump if not "/"
0214	E0DE	7D 00 91		TST	TMP1	
0215	E0E1	26 08		BNE	COMM3	jump if not enabled
0216	E0E3	D7 7E		STAB	COUNT	
0217	E0E5	CE E5 0D		LDX	#MSLASH	
0218	E0E8	7E E1 32		JMP	EXEC	execute "/"
0219	E0EB	CE E4 AF	COMM3	LDX	#MSG8	"command?"
0220	E0EE	BD E2 08		JSR	OUTSTRG	
0221	E0F1	7E E0 3E		JMP	MAIN	
0222	E0F4	BD E2 7F	COMM4	JSR	HEXBIN	
0223	E0F7	20 BB		BRA	COMM1	
0224						
0225						
0226						
0227						
0228						
0229						
0230						
0231						
0232						
0233						
0234						
0235	E0F9	D7 7E	SRCH	STAB	COUNT	size of command entered
0236	E0FB	CE E4 27		LDX	#COMTABL	pointer to table
0237	E0FE	DF 83		STX	PTR1	pointer to next entry
0238	E100	DE 83	SRCH1	LDX	PTR1	
0239	E102	18 CE 00 60		LDY	#COMBUFF	pointer to command buffer
0240	E106	E6 00		LDAB	0,X	
0241	E108	C1 FF		CMPB	#\$FF	
0242	E10A	26 09		BNE	SRCH2	
0243	E10C	CE E4 91		LDX	#MSG2	"command not found"
0244	E10F	BD E2 08		JSR	OUTSTRG	
0245	E112	7E E0 3E		JMP	MAIN	
0246	E115	3C	SRCH2	PSHX		compute next table entry
0247	E116	CB 03		ADDB	#\$3	
0248	E118	3A		ABX		
0249	E119	DF 83		STX	PTR1	
0250	E11B	38		PULX		
0251	E11C	5F		CLRB		
0252	E11D	5C	SRCHLP	INCB		match characters loop
0253	E11E	A6 01		LDAA	1,X	read table
0254	E120	18 A1 00		CMPA	0,Y	compare to combuff
0255	E123	26 DB		BNE	SRCH1	try next entry

* Search tables for command. At this point,
 * COMBUFF holds the command field to be executed,
 * and B = # of characters in the command field.
 * The command table holds the whole command name
 * but only the first n characters of the command
 * must match what is in COMBUFF where n is the
 * number of characters entered by the user.

```

0256 E125 08          INX          move pointers
0257 E126 18 08      INY
0258 E128 D1 7E      CMPB COUNT
0259 E12A 2D F1      BLT  SRCHLP    loop count 1 times
0260 E12C DE 83      LDX  PTR1
0261 E12E 09          DEX
0262 E12F 09          DEX
0263 E130 EE 00      LDX  0,X        jump address from table
0264 E132 AD 00      EXEC JSR  0,X        call task as subroutine
0265 E134 7E E0 3E    JMP  MAIN
0266
0267 *****
0268 * INCHAR() - this routine waits for input
0269 *   from the SCI or SPI
0270 *****
0271
0272 E137 BD E1 41    INCHAR JSR  INPUT
0273 E13A 4D          TSTA
0274 E13B 27 FA      BEQ  INCHAR
0275 E13D BD E1 76    JSR  OUTPUT    echo character in (A)
0276 E140 39          INCHAR10 RTS
0277
0278 *****
0279 * INPUT() - this routine calls the specific
0280 * routine to read the SPI or SCI. It also
0281 * disarms the COP. The character is
0282 * returned in register A.
0283 *****
0284
0285 E141 86 55      INPUT LDAA #55
0286 E143 B7 10 3A  STAA COPRST
0287 E146 86 AA      LDAA #5AA
0288 E148 B7 10 3A  STAA COPRST
0289 E14B 96 01      LDAA IODEV
0290 E14D 26 04      BNE  INPUT1
0291 E14F BD E1 57    JSR  INSCI
0292 E152 39          RTS
0293 E153 BD F9 EB    INPUT1 JSR  VIDEO
0294 E156 39          RTS
0295
0296 *****
0297 * INSCI() - get input from the SCI. Data received
0298 * is returned in register A.
0299 *****
0300
0301 E157 B6 10 2E    INSCI LDAA SCSR
0302 E15A 84 20      ANDA #520
0303 E15C 27 05      BEQ  INSCI1
0304 E15E B6 10 2F    LDAA SCDAT
0305 E161 84 7F      ANDA #57F
0306 E163 39          INSCI1 RTS

```

```

0307
0308 *****
0309 * CRTBKSPC() - this routine is used when a backspace
0310 * character is received from the SPI. Updates the
0311 * CHARPTR and erases the appropriate data in the
0312 * CHARBUFF. On entry X points to next free
0313 * location in CHARBUFF. X-1 points to a backspace
0314 * character, and X-2 has the data to be erased
0315 * Destroys: A,B,X
0316 *****
0317
0318 E164 86 20 CRTBKSPC LDAA #SPACE
0319 E166 09 DEX decrement pointer
0320 E167 A7 00 STAA 0,X put space in previous loc
0321 E169 CB FE ADDB #-2 decrement offset by two
0322 E16B 27 05 BEQ FACKSP1
0323 E16D 09 DEX
0324 E16E A7 00 STAA 0,X put space in old location
0325 E170 20 01 BRA BACKSP2
0326 E172 5C BACKSP1 INCB
0327 E173 D7 02 BACKSP2 STAB CHARPTR update pointer
0328 E175 39 RTS
0329
0330 *****
0331 * OUTPUT(a) - this routine outputs the character in
0332 * A to the either the SCI or SPI. On exit, A still has
0333 * the character.
0334 *****
0335
0336 E176 37 OUTPUT PSHB
0337 E177 D6 01 LDAB IODEV
0338 E179 26 05 BNE OUTPUT1
0339 E17B BD E3 DB JSR OUTSCI
0340 E17E 20 03 BRA OUTPUT4
0341 E180 BD E1 D5 OUTPUT1 JSR OUTCRT
0342 E183 33 OUTPUT4 PULB
0343 E184 39 RTS
0344
0345 *****
0346 * initcrt() - this routine initializes the
0347 * SPI and configures port D. Port D must
0348 * be configured before the SPI is initialized
0349 * or the SPI will lock up, thinking it has had
0350 * a write collision.
0351 *****
0352 * initialize DDRD
0353 E185 86 3A INITCRT LDAA #DDRDINIT
0354 E187 B7 10 09 STAA DDRD
0355 * initialize D5 to a logic one
0356 E18A B6 10 08 LDAA PORTD
0357 E18D 8A 20 ORAA #SSHIGH set bit 5 to ZERO

```

```

0358 E18F B7 10 08          STAA PORTD
0359                      * Configure the SPCR
0360 E192 86 5C          LDAA #CTRLINIT      Bit pattern into SPCR
0361 E194 B7 10 28          STAA SPCR
0362 E197 39              RTS
0363
0364                      *****
0365                      * crtchk() - this routine checks for a key depression
0366                      * via the SPI. If a key is depressed then A is returned
0367                      * with the value 1, else A is 0.
0368                      *****
0369
0370 E198 86 FE          CRTCHK LDAA #$FE          first pattern
0371 E19A F6 10 29          LDAB SPSR          clear spi flags
0372 E19D B7 10 2A          STAA SPDR          send first pattern
0373 E1A0 36              PSHA          save A
0374 E1A1 3D              MUL          kill time
0375 E1A2 32              PULA          restore A
0376 E1A3 BD E2 63          JSR HORPULSE      latch data
0377 E1A6 F6 10 29          CRTCHK0 LDAB SPSR
0378 E1A9 0D              SEC
0379 E1AA 49              ROLA
0380 E1AB 24 12          BCC CRTCHK1          all patterns have been sent
0381 E1AD B7 10 2A          STAA SPDR
0382 E1B0 36              PSHA          save A
0383 E1B1 3D              MUL          kill time
0384 E1B2 32              PULA          restore A
0385 E1B3 BD E2 63          JSR HORPULSE
0386 E1B6 F6 10 2A          LDAB SPDR
0387 E1B9 C1 FF          CMPB #$FF
0388 E1BB 26 10          BNE CRTCHK2
0389 E1BD 20 E7          BRA CRTCHK0
0390 E1BF B7 10 2A          CRTCHK1 STAA SPDR          need to check last pattern
0391 E1C2 3D              MUL          kill time
0392 E1C3 3D              MUL          kill time
0393 E1C4 F6 10 2A          LDAB SPDR          get last pattern
0394 E1C7 C1 FF          CMPB #$FF
0395 E1C9 26 02          BNE CRTCHK2
0396 E1CB 4F              CLRA
0397 E1CC 39              RTS          return with a=0; no key
0398 E1CD 81 00          CRTCHK2 CMPA #0          check for nothing connected
0399 E1CF 26 01          BNE CRTCHK3
0400 E1D1 39              RTS          returns with a=0; no key
0401 E1D2 86 01          CRTCHK3 LDAA #1
0402 E1D4 39              RTS          key depressed
0403
0404                      *****
0405                      * OUTCRT(a) - this routine is called if the SPI is the
0406                      * external device. The character in A is put in the
0407                      * CHARBUFF and the CHARPTR is updated. If there is
0408                      * no room in CHARBUFF, then the routine calls VIDEO

```

```

0409      * which will display the contents of CHARBUFF and wait
0410      * for the user to hit any key. The CHARBUFF will be emptied
0411      * and the data in A placed in CHARBUFF. A still contains the
0412      * character data upon exit. Destroys B.
0413      *****
0414
0415      E1D5  3C      OUTCRT      PSHX      save (X)
0416      E1D6  D6 02      LDAB  CHARPTR  get pointer
0417      E1D8  C1 E1      CMPB  #BUFFLEN  buffer full?
0418      E1DA  27 0B      BEQ   OUTCRT1    yes-display data
0419      E1DC  CE 00 05      LDX   #CHARBUFF  no-put data in buffer
0420      E1DF  3A      ABX
0421      E1E0  A7 00      STAA  0,X      put character in buffer
0422      E1E2  7C 00 02      INC   CHARPTR
0423      E1E5  20 0E      BRA   OUTCRT2
0424      E1E7  36      OUTCRT1    PSHA      save (A)
0425      E1E8  BD F9 EB      JSR   VIDEO    display data;wait for keyentry
0426      E1EB  BD E2 4E      JSR   CLRBUFF  clean out buffer
0427      E1EE  C6 01      LDAB  #1      reset pointer
0428      E1F0  D7 02      STAB  CHARPTR  save it
0429      E1F2  32      PULA      get character
0430      E1F3  97 05      STAA  CHARBUFF  put character in buffer
0431      E1F5  38      OUTCRT2    PULX      restore (X)
0432      E1F6  39      RTS
0433
0434      *****
0435      * OUTCRLF() - output a carriage return and linefeed.
0436      * Returns A containing an ASCII carriage return.
0437      *****
0438
0439      E1F7  7D 00 01      OUTCRLF    TST   IODEV
0440      E1FA  26 0B      BNE   OUTCRLF1
0441      E1FC  86 0D      LDAA  #CRETURN
0442      E1FE  BD E1 76      JSR   OUTPUT
0443      E201  4F      CLRA
0444      E202  BD E1 76      JSR   OUTPUT
0445      E205  86 0D      LDAA  #CRETURN
0446      E207  39      OUTCRLF1    RTS
0447
0448      *****
0449      * OUTSTRG() - this routine calls an SPI or SCI routine
0450      * to output a string of characters.
0451      *****
0452
0453      E208  BD E1 F7      OUTSTRG    JSR   OUTCRLF
0454      E20B  36      OUTSTRG0    PSHA
0455      E20C  96 01      LDAA  IODEV
0456      E20E  26 0E      BNE   OUTSTRG1
0457      E210  BD E2 1A      JSR   SCISTRG
0458      E213  20 03      BRA   OUTSTRG2
0459      E215  BD E2 35      OUTSTRG1    JSR   CRTSTRG

```

```

0460 E218 32      OUTSTRG2  PULA
0461 E219 39      RTS
0462
0463 *****
0464 *SCISTRG() - output a string of ASCII bytes starting at x
0465 * until an end of text to the SCI. X is updated to the EOT
0466 * character.
0467 *****
0468
0469 E21A A6 00      SCISTRG   LDAA 0,X
0470 E21C 81 04      CMPA #EOT
0471 E21E 27 14      BEQ  SCISTRG2
0472 E220 BD E1 76   JSR  OUTPUT
0473 E223 08        INX
0474 E224 BD E1 41   JSR  INPUT
0475 E227 27 F1      BEQ  SCISTRG
0476 E229 81 17      CMPA #CTLW
0477 E22B 26 ED      BNE  SCISTRG
0478 E22D BD E1 41   SCISTRG1 JSR  INPUT
0479 E230 27 FB      BEQ  SCISTRG1
0480 E232 20 E6      BRA  SCISTRG
0481 E234 39      SCISTRG2  RTS
0482
0483 *****
0484 * CRTSTRG()- output string of ASCII bytes starting at x until
0485 * end of text to SPI. X is updated to the eot character.
0486 * It then will continue send characters to output() until the eot
0487 * is reached. The routine will then wait for any key to be
0488 * depressed by the user. Destroys A.
0489 *****
0490
0491 E235 BD E2 4E   CRTSTRG   JSR  CLRBUFF   clear the character buffer
0492 E238 A6 00      CRTSTRG0  LDAA 0,X      get character
0493 E23A 81 04      CMPA #EOT   end of text?
0494 E23C 27 09      BEQ  CRTSTRG1 yes - display message
0495 E23E BD E2 74   JSR  UPCASE   make sure it is upper case
0496 E241 BD E1 76   JSR  OUTPUT   put character in buffer
0497 E244 08        INX        point to next character
0498 E245 20 F1      BRA  CRTSTRG0 get next character
0499 E247 BD E2 41   CRTSTRG1  JSR  INPUT   wait for input from user
0500 E24A BD E2 4E   JSR  CLRBUFF   clean out the buffer
0501 E24D 39      RTS
0502 *****
0503 * clrbuff() - This routine fills CHARBUFF with ASCII space
0504 * characters and resets CHARPTR to zero.
0505 *****
0506
0507 E24E 3C      CLRBUFF   PSHX      save registers
0508 E24F 36      PSHA
0509 E250 CE 00 05  LDX  #CHARBUFF
0510 E253 86 20      LDAA #S20

```



```

0511 E255 A7 00      ANOTHER   STAA 0,X
0512 E257 08                INX
0513 E258 8C 00 23      CPX    #CHARBUFF+BUFFLEN  buffer clear?
0514 E25B 26 F8                BNE    ANOTHER
0515 E25D 7F 00 02      CLR    CHARPTR          reset pointer
0516 E260 32                PULA                restore registers
0517 E261 38                PULX
0518 E262 39                RTS
0519
0520                *****
0521                * horpulse() - this routine outputs a pulse on the D5 pin.
0522                * The pulse width is 14 E clocks (7  $\mu$ s with 2 mhz E clock)
0523                * Destroys B.
0524                *****
0525 E263 F6 10 08      HORPULSE  LDAB PORTD
0526 E266 C4 DF                ANDB                #SSLOW
0527 E268 F7 10 08      STAB PORTD
0528 F26B 8F                XGDX                kill time
0529 E26C 8F                XGDX                kill time
0530 E26D 01                NOP                kill time
0531 E26E CA 20                ORAB                #SSHIGH
0532 E270 F7 10 08      STAB PORTD
0533 E273 39                RTS
0534
0535                *****
0536                * UPCASE(a) - If the contents of A is alpha,
0537                * returns a converted to uppercase.
0538                *****
0539
0540 E274 81 61      UPCASE      CMPA #'a'
0541 E276 2D 06                BLT    UPCASE1      jump if < a
0542 E278 81 7A                CMPA #'z'
0543 E27A 2E 02                BGT    UPCASE1      jump if > z
0544 E27C 80 20                SUBA  #$20      convert
0545 E27E 39      UPCASE1      RTS
0546
0547                *****
0548                * HEXBIN(a) - Convert the ASCII character in a
0549                * to binary and shift into shftreg. Returns value
0550                * in tmp1 incremented if a is not hex.
0551                *****
0552
0553 E27F 36      HEXBIN      PSHA
0554 E280 37                PSHB
0555 E281 3C                PSHX
0556 E282 BD E2 74      JSR    UPCASE      convert to upper case
0557 E285 81 30                CMPA #'0'
0558 E287 2D 22                BLT    HEXNOT      jump if a < $30
0559 E289 81 39                CMPA #'9'
0560 E28B 2F 0A                BLE    HEXNMB      jump if 0-9
0561 E28D 81 41                CMPA #'A'

```

0562	E28F	2D 1A		BLT	HEXNOT	jump if \$39 > a < \$41
0563	E291	81 46		CMPA	#F	
0564	E293	2E 16		BGT	HEXNOT	jump if a > \$46
0565	E295	8B 09		ADDA	#\$9	convert \$A-\$F
0566	E297	84 0F	HEXNMB	ANDA	#\$0F	convert to binary
0567	E299	CE 00 68		LDX	#SHFTREG	
0568	E29C	C6 04		LDAB	#4	
0569	E29E	68 01	HEXSHFT	ASL	1,X	2 byte shift through
0570	E2A0	69 00		ROL	0,X	carry bit
0571	E2A2	5A		DECB		
0572	E2A3	2E F9		BGT	HEXSHFT	shift 4 times
0573	E2A5	AA 01		ORAA	1,X	
0574	E2A7	A7 01		STAA	1,X	
0575	E2A9	20 03		BRA	HEXRTS	
0576	E2AB	7C 00 91	HEXNOT	INC	TMP1	indicate not hex
0577	E2AE	38	HEXRTS	PULX		
0578	E2AF	33		PULB		
0579	E2B0	32		PULA		
0580	E2B1	39		RTS		

0581

0582

0583

0584

0585

0586

0587

0588

0589

0590

0591

0592

0593

0594

0595

0596

0597

0598

0599

0600

0601

0602

0603

0604

0605

0606

0607

0608

0609

0610

0611

0612

* BUFFARG() - Build a hex argument from the
 * contents of the input buffer. Characters are
 * converted to binary and shifted into shftreg
 * until a non-hex character is found. On exit
 * shftreg holds the last four digits read, count
 * holds the number of digits read, ptrbuff points
 * to the first non-hex character read, and A holds
 * that first non-hex character.

BUFFARG

CLR TMP1

not hex indicator

CLR COUNT

or digits

CLR SHFTREG

CLR SHFTREG+1

JSR WSKIP

JSR RDBUFF

read char

JSR HEXBIN

TST TMP1

BNE BUFFRTS

jump if not hex

INC COUNT

JSR INCBUFF

move buffer pointer

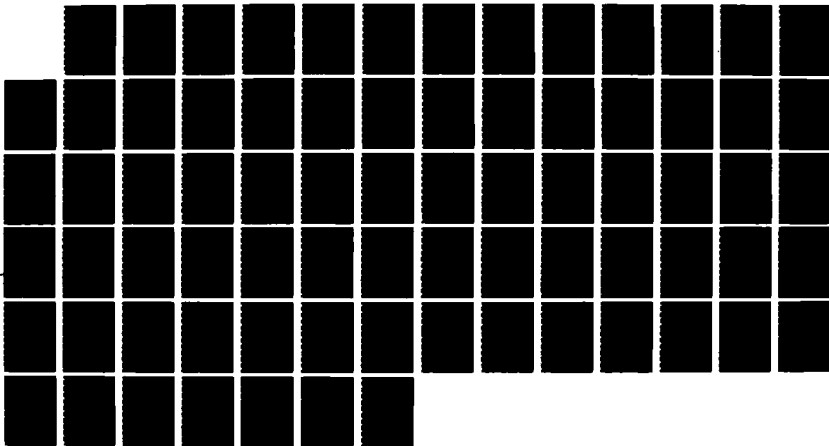
BRA BUFFLP

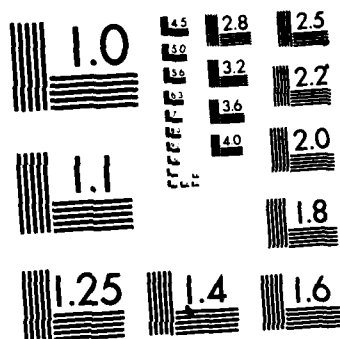
RTS

BUFFRTS

* TERMARG() - Build a hex argument from the
 * terminal. Characters are converted to binary
 * and shifted into shftreg until a non-hex character
 * is found. On exit shftreg holds the last four
 * digits read, count holds the number of digits

AD-A172 963 A LOW COST VIDEO DISPLAY SYSTEM USING THE MOTOROLA 6811 2/2
SINGLE-CHIP MICROCOMPUTER(U) AIR FORCE INST OF TECH
WRIGHT-PATTERSON AFB OH K E WILLIAMS AUG 86
UNCLASSIFIED AFIT/CI/NR-86-179T F/G 9/2 NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

```

0613          * read, and A holds the first non-hex character.
0614          *****
0615
0616 E2D5 7F 00 7E  TERMARG  CLR  COUNT
0617 E2D8 7F 00 68          CLR  SHFTREG
0618 E2DB 7F 00 69          CLR  SHFTREG+1
0619 E2DE BD E1 37  TERM0    JSR  INCHAR
0620 E2E1 81 18          CMPA  #CTLX
0621 E2E3 27 04          BEQ  TERM1      jump if controlx
0622 E2E5 81 7F          CMPA  #DEL
0623 E2E7 26 03          BNE  TERM2      jump if not delete
0624 E2E9 7E E0 3E  TERM1    JMP  MAIN      abort
0625 E2EC 7F 00 91  TERM2    CLR  TMP1      hex indicator
0626 E2EF BD E2 7F          JSR  HEXBIN
0627 E2F2 7D 00 91          TST  TMP1
0628 E2F5 26 05          BNE  TERM3      jump if not hex
0629 E2F7 7C 00 7E          INC  COUNT
0630 E2FA 20 E2          BRA  TERM0
0631 E2FC 39          TERM3    RTS
0632
0633          *****
0634          * CHGBYT() - If shftreg is not empty, put
0635          * contents of shftreg at address in X. If X
0636          * is an address in EEPROM then program it.
0637          *****
0638
0639 E2FD 7D 00 7E  CHGBYT    TST  COUNT
0640 E300 27 60          BEQ  CHGBYT4      jump if shftreg empty
0641 E302 96 69          LDAA  SHFTREG+1
0642 E304 A7 00  CHGBYT0    STAA  0,X      attempt to write
0643 E306 A6 00          LDAA  0,X
0644 E308 91 69          CMPA  SHFTREG+1
0645 E30A 27 45          BEQ  CHGBYT3      jump if it worked
0646 E30C 8C 10 3F          CPX  #CONFIG
0647 E30F 27 0A          BEQ  CHGBYT1      jump if config reg
0648 E311 8C B6 00          CPX  #$B600
0649 E314 25 3B          BLO  CHGBYT3      jump if not EE
0650 E316 8C B7 FF          CPX  #$B7FF
0651 E319 22 36          BHI  CHGBYT3      jump if not EE
0652 E31B A6 00  CHGBYT1    LDAA  0,X
0653 E31D 81 FF          CMPA  #$FF
0654 E31F 27 15          BEQ  CHGBYT2      jump if already erased
0655 E321 86 16          LDAA  #$16      do byte erase
0656 E323 B7 10 3B          STAA  PPROG
0657 E326 86 FF          LDAA  #$FF
0658 E328 A7 00          STAA  0,X
0659 E32A 86 17          LDAA  #$17
0660 E32C B7 10 3B          STAA  PPROG
0661 E32F 8D 32          BSR  CHGWAIT
0662 E331 86 00          LDAA  #$00
0663 E333 B7 10 3B          STAA  PPROG      end of byte erase

```

```

0664 E336 96 69      CHGBYT2    LDAA SHFTREG+1
0665 E338 81 FF      CMPA #$FF
0666 E33A 27 15      BEQ  CHGBYT3    jump if no need to program
0667 E33C 86 02      LDAA #$02      do byte program
0668 E33E B7 10 3B    STAA PPROG
0669 E341 96 69      LDAA SHFTREG+1
0670 E343 A7 00      STAA 0,X
0671 E345 86 03      LDAA #$03
0672 E347 B7 10 3B    STAA PPROG
0673 E34A 8D 17      BSR  CHGWAIT
0674 E34C 86 00      LDAA #$00
0675 E34E B7 10 3B    STAA PPROG      end of byte program
0676 E351 A6 00      CHGBYT3    LDAA 0,X
0677 E353 91 69      CMPA SHFTREG+1
0678 E355 27 0B      BEQ  CHGBYT4
0679 E357 3C          PSHX
0680 E358 CE E4 AA    LDX  #MSG6      "rom"
0681 E35B BD E2 08    JSR  OUTSTRG
0682 E35E BD E1 F7    JSR  OUTCRLF
0683 E361 38          PULX
0684 E362 39          CHGBYT4    RTS
0685
0686 E363 3C          CHGWAIT    PSHX      delay 10 ms
0687 E364 CE 0D AC    LDX  #3500
0688 E367 09          CHGWAIT1   DEX
0689 E368 26 FD      BNE  CHGWAIT1
0690 E36A 38          PULX
0691 E36B 39          RTS
0692
0693 *****
0694 * BPCLR() - Clear all entries in the
0695 * table of breakpoints.
0696 *****
0697
0698 E36C CE 00 6A      BPCLR      LDX  #BRKTABL
0699 E36F C6 08          LDAB #8
0700 E371 6F 00          BPCLR1     CLR  0,X
0701 E373 08            INX
0702 E374 5A            DECB
0703 E375 2E FA          BGT  BPCLR1     loop 8 times
0704 E377 39            RTS
0705
0706 *****
0707 * RDBUFF() - Read the character in CHARBUFF
0708 * pointed at by ptrbuff into A. Returns ptrbuff
0709 * unchanged.
0710 *****
0711
0712 E378 3C          RDBUFF      PSHX
0713 E379 DE 81          LDX  PTR0
0714 E37B A6 00          LDAA 0,X

```

```

0715 E37D 38 PULX
0716 E37E 39 RTS
0717
0718 *****
0719 * INCBUFF(), DECBUFF() - Increment or decrement
0720 * ptrbuff.
0721 *****
0722
0723 E37F 3C INCBUFF PSHX
0724 E380 DE 81 LDX PTR0
0725 E382 08 INX
0726 E383 20 04 BRA INCDEC
0727 E385 3C DECBUFF PSHX
0728 E386 DE 81 LDX PTR0
0729 E388 09 DEX
0730 E389 DF 81 INCDEC STX PTR0
0731 E38B 38 PULX
0732 E38C 39 RTS
0733
0734 *****
0735 * WSKIP() - Read from the CHARBUFF until a
0736 * non whitespace (space, comma, tab) character
0737 * is found. Returns ptrbuff pointing to the
0738 * first non-whitespace character and a holds
0739 * that character.
0740 *****
0741
0742 E38D BD E3 78 WSKIP JSR RDBUFF read character
0743 E390 BD E3 9B JSR WCHEK
0744 E393 26 05 BNE WSKIP1 jump if not wspc
0745 E395 BD E3 7F JSR INCBUFF move pointer
0746 E398 20 F3 BRA WSKIP loop
0747 E39A 39 WSKIP1 RTS
0748
0749 *****
0750 * WCHEK(a) - Returns z=1 if a holds a
0751 * whitespace character, else z=0.
0752 *****
0753 E39B 81 2C WCHEK CMPA #$2C comma
0754 E39D 27 06 BEQ WCHEK1
0755 E39F 81 20 CMPA #$20 space
0756 E3A1 27 02 BEQ WCHEK1
0757 E3A3 81 09 CMPA #$09 tab
0758 E3A5 39 WCHEK1 RTS
0759
0760 *****
0761 * DCHEK(a) - Returns Z=1 if a = whitespace
0762 * or carriage return or period. Else returns z=0.
0763 *****
0764
0765 E3A6 BD E3 9B DCHEK JSR WCHEK

```

```

0766 E3A9 27 06          BEQ  DCHEK1      jump if whitespace
0767 E3AB 81 0D          CMPA  #$0D
0768 E3AD 27 02          BEQ  DCHEK1
0769 E3AF 81 2E          CMPA  #'.'
0770 E3B1 39             DCHEK1      RTS
0771
0772 *****
0773 * CHKABRT() - Checks for a control x or delete
0774 * from the terminal. If found, the stack is
0775 * reset and the control is transferred to main.
0776 * Note that this is an abnormal termination.
0777 * If the input from the terminal is a control W
0778 * then this routine keeps waiting until any other
0779 * character is read.
0780 *****
0781
0782 E3B2 BD E1 41      CHKABRT      JSR  INPUT
0783 E3B5 27 14          BEQ  CHK4      jump if no input
0784 E3B7 81 17          CMPA  #CTLW
0785 E3B9 26 05          BNE  CHK2      jump in not cntlw
0786 E3BB BD E1 41      CHKABRT1     JSR  INPUT
0787 E3BE 27 FB          BEQ  CHKABRT1  jump if no input
0788 E3C0 81 7F          CHK2         CMPA  #DEL
0789 E3C2 27 04          BEQ  CHK3      jump if delete
0790 E3C4 81 18          CMPA  #CTLX
0791 E3C6 26 03          BNE  CHK4      jump if not control x
0792 E3C8 7E E0 3E      CHK3         JMP  MAIN      abort
0793 E3CB 39             CHK4         RTS              return
0794
0795 *****
0796 * onsci() - initialize the SCI for 600 baud
0797 * at 8 mhz extal.
0798 *****
0799 E3CC 86 34          ONSCI        LDAA  #$34
0800 E3CE B7 10 2B      STAA  BAUD
0801 E3D1 4F            CLRA
0802 E3D2 B7 10 2C      STAA  SCCR1
0803 E3D5 86 0C          LDAA  #$0C
0804 E3D7 B7 10 2D      STAA  SCCR2
0805 E3DA 39            RTS
0806
0807 *****
0808 * OUTSCI() - Output A to sci. IF autolf = 1,
0809 *          cr and lf sent as crlf.
0810 *****
0811
0812 E3DB 7D 00 7D      OUTSCI        TST  AUTOLF
0813 E3DE 27 10          BEQ  OUTSCI2    jump if autolf=0
0814 E3E0 8D 0E          BSR  OUTSCI2
0815 E3E2 81 0D          CMPA  #$0D
0816 E3E4 26 04          BNE  OUTSCI1

```



```

0817 E3E6 86 0A          LDAA #$0A          if cr, send lf
0818 E3E8 20 06          BRA OUTSCI2
0819 E3EA 81 0A          OUTSCI1 CMPA #$0A
0820 E3EC 26 0E          BNE OUTSCI3
0821 E3EE 86 0D          LDAA #$0D          if lf, send cr
0822 E3F0 F6 10 2E       OUTSCI2 LDAB SCSR          read status
0823 E3F3 C5 80          BITB #$80
0824 E3F5 27 F9          BEQ OUTSCI2          loop until tdre=1
0825 E3F7 84 7F          ANDA #$7F          mask parity
0826 E3F9 B7 10 2F       STAA SCDAT          send character
0827 E3FC 39          OUTSCI3 RTS
0828
0829 *****
0830 * OUTRHLF(), OUTLHLF(), OUTA()
0831 *Convert A from binary to ASCII and output.
0832 *Contents of A are destroyed..
0833 *****
0834
0835 E3FD 44          OUTLHLF LSRA          shift data to right
0836 E3FE 44          LSRA
0837 E3FF FF          LSRA
0838 E400 44          LSRA
0839 E401 84 0F       OUTRHLF ANDA #$0F          mask top half
0840 E403 8B 30       ADDA #$30          convert to ascii
0841 E405 81 39       CMPA #$39
0842 E407 2F 02       BLE OUTA          jump if 0-9
0843 E409 8B 07       ADDA #$07          convert to hex A-F
0844 E40B BD E1 76   OUTA JSR OUTPUT          output character
0845 E40E 39          RTS
0846
0847 *****
0848 * OUT1BYT(x) - Convert the byte at X to two
0849 *ASCII characters and output. Return X pointing
0850 *to next byte.
0851 *****
0852
0853 E40F 36          OUT1BYT PSHA
0854 E410 A6 00       LDAA 0,X          get data in a
0855 E412 36          PSHA          save copy
0856 E413 8D E8       BSR OUTLHLF          output left half
0857 E415 32          PULA          retrieve copy
0858 E416 8D E9       BSR OUTRHLF          output right half
0859 E418 32          PULA
0860 E419 08          INX
0861 E41A 39          RTS
0862
0863 *****
0864 * OUT1BSP(x), OUT2BSP(x) - Output 1 or 2 bytes
0865 *at x followed by a space. Returns x pointing to
0866 *next byte.
0867 *****

```

```

0868
0869 E41B BD E4 0F OUT2BSP JSR OUT1BYT do first byte
0870 E41E BD E4 0F OUT1BSP JSR OUT1BYT do next byte
0871 E421 86 20 OUTSPAC LDAA #$20 output a space
0872 E423 BD E1 76 JSR OUTPUT
0873 E426 39 RTS
0874

```

*** COMMAND TABLE ***

```

0875
0876 E427 COMTABL EQU *
0877 E427 03 FCB 3
0878 E428 41 53 4D FCC 'ASM'
0879 E42B EE EE FDB #ASSEM
0880 E42D 02 FCB 2
0881 E42E 42 52 FCC 'BR'
0882 E430 EC B2 FDB #BREAK
0883 E432 04 FCB 4
0884 E433 42 55 4C 4B FCC 'BULK'
0885 E437 EA FD FDB #BULK
0886 E439 07 FCB 7
0887 E43A 42 55 4C 4B 41 4C 4C FCC 'BULKALL'
0888 E441 EB 02 FDB #BULKALL
0889 E443 04 FCB 4
0890 E444 43 41 4C 4C FCC 'CALL'
0891 E448 EB 2F FDB #CALL
0892 E44A 02 FCB 2
0893 E44B 4D 44 FCC 'MD'
0894 E44D E6 29 FDB #DUMP
0895 E44F 01 FCB 1
0896 E450 47 FCC 'G'
0897 E451 EB 66 FDB #GO
0898 E453 02 FCB 2
0899 E454 45 58 FCC 'EX'
0900 E456 EB 89 FDB #EX
0901 E458 04 FCB 4
0902 E459 48 45 4C 50 FCC 'HELP'
0903 E45D E7 CF FDB #HELP
0904 E45F 02 FCB 2
0905 E460 4D 4D FCC 'MM'
0906 E462 E4 F5 FDB #MEMORY
0907 E464 02 FCB 2
0908 E465 4D 56 FCC 'MV'
0909 E467 E5 7A FDB #MOVE
0910 E469 01 FCB 1
0911 E46A 50 FCC 'P'
0912 E46B EB A8 FDB #PROCEED
0913 E46D 02 FCB 2
0914 E46E 52 44 FCC 'RD'
0915 E470 E7 5B FDB #REGISTER
0916 E472 01 FCB 1
0917 E473 54 FCC 'T'
0918 E474 EB AF FDB #TRACE

```

LENGTH OF COMMAND
ASCII COMMAND
COMMAND ADDRESS

```

0919 E476 04 FCB 4
0920 E477 4C 4F 41 44 FCC 'LOAD'
0921 E47B ED 69 FDB #LOAD
0922 E47D 04 FCB 4
0923 E47E 56 45 52 46 FCC 'VERF'
0924 E482 ED 61 FDB #VERIFY
0925 E484 FF FCB -1 end of table
0926
0927 *** TEXT TABLES ***
0928 E485 42 55 46 46 MSG1 FCC 'BUFFALO 2.2'
0929 E490 04 FCB EOT
0930 E491 57 68 61 74MSG2 FCC 'What?'
0931 E496 04 FCB EOT
0932 E497 54 6F 6F 20 MSG3 FCC 'Too Long'
0933 E49F 04 FCB EOT
0934 E4A0 46 76 6C 6CMSG4 FCC 'Full'
0935 E4A4 04 FCB EOT
0936 E4A5 4F 70 2D 20MSG5 FCC 'Op-'
0937 E4A9 04 FCB EOT
0938 E4AA 72 6F 6D 2DMSG6 FCC 'rom-'
0939 E4AE 04 FCB EOT
0940 E4AF 43 6F 6D 6DMSG8 FCC 'Command?'
0941 E4B7 04 FCB EOT
0942 E4B8 42 61 64 20 MSG9 FCC 'Bad argument'
0943 E4C4 04 FCB EOT
0944 E4C5 64 6F 6E 65 MSG11 FCC 'done'
0945 E4C9 04 FCB EOT
0946 E4CA 63 68 65 63 MSG12 FCC 'checksum error'
0947 E4D8 04 FCB EOT
0948 E4D9 65 72 72 6F MSG13 FCC 'error addr'
0949 E4E4 04 FCB EOT
0950 E4E5 4E 4F 54 20 MSG14 FCC 'NOT IN SPI MODE'
0951 E4F4 04 FCB EOT
0952
0953 *****
0954 * MEMORY [<addr>]
0955 * [<addr>]/
0956 * Opens memory and allows user to modify the
0957 * contents at <addr> or the last opened location.
0958 * Subcommands:
0959 * [<data>]<cr> - Close current location and OPEN NEXT.
0960 * [<data>]<-> - Close current and open previous.
0961 * [<data>]<.> - Close current and EXIT.
0962 * [<data>]<=> - Reopen current location.
0963 * The contents of the current location is only
0964 * changed if valid data is entered before each
0965 * subcommand.
0966 *****
0967
0968 E4F5 BD E3 8D MEMORY JSR WSKIP
0969 E4F8 81 0D CMPA #SD

```

0970	E4FA	27 1A		BEQ	MEM1	jump if cr
0971	E4FC	BD E2 B2		JSR	BUFFARG	
0972	E4FF	BD E3 8D		JSR	WSKIP	
0973	E502	81 0D		CMPA	#\$D	
0974	E504	27 07		BEQ	MSLASH	jump if cr
0975	E506	CE E4 B8		LDX	#MSG9	"bad argument"
0976	E509	BD E2 08		JSR	OUTSTRG	
0977	E50C	39		RTS		
0978	E50D	7A 00 7E	MSLASH	DEC	COUNT	
0979	E510	27 04		BEQ	MEM1	jump if no argument
0980	E512	DE 68		LDX	SHFTREG	
0981	E514	DF 7F		STX	PTRMEM	update "current location"
0982	E516	BD E1 F7	MEM1	JSR	OUTCRLF	
0983	E519	BD E2 4E		JSR	CLRBUFF	if dev = SPI then clr buffer
0984	E51C	CE 00 7F	MEM2	LDX	#PTRMEM	
0985	E51F	BD E4 1B		JSR	OUT2BSP	output address
0986	E522	DE 7F	MEM3	LDX	PTRMEM	
0987	E524	BD E4 21		JSR	OUTSPAC	
0988	E527	BD E4 1E		JSR	OUT1BSP	output contents
0989	E52A	C6 03		LDAB	#3	
0990	E52C	BD E4 21	MEM35	JSR	OUTSPAC	output 3 spaces
0991	E52F	5A		DECB		
0992	E530	26 FA		BNE	MEM35	
0993	E532	86 3E		LDAA	#PROMPT	output prompt character
0994	E534	BD E1 76		JSR	OUTPUT	
0995	E537	7F 00 68		CLR	SHFTREG	
0996	E53A	7F 00 69		CLR	SHFTREG+1	
0997	E53D	BD E2 D5	MEM4	JSR	TERMARG	
0998	E540	BD E2 74		JSR	UPCASE	
0999	E543	DE 7F		LDX	PTRMEM	
1000	E545	81 0D		CMPA	#CRETURN	
1001	E547	27 15		BEQ	MEMSP	jump if CR
1002	E549	81 2D		CMPA	#'-'	
1003	E54B	27 1A		BEQ	MEMUA	jump if MINUS SIGN
1004	E54D	81 3D		CMPA	#'='	
1005	E54F	27 1F		BEQ	MEMSL	jump if =
1006	E551	81 2E		CMPA	#'.'	
1007	E553	27 21		BEQ	MEMCR	jump if PERIOD
1008	E555	CE E4 AF		LDX	#MSG8	"command?"
1009	E558	BD E2 08		JSR	OUTSTRG	
1010	E55B	7E E5 16		JMP	MEM1	
1011	E55E	BD E2 FD	MEMSP	JSR	CHGBYT	
1012	E561	08		INX		
1013	E562	DF 7F		STX	PTRMEM	
1014	E564	7E E5 16		JMP	MEM1	output contents
1015	E567	BD E2 FD	MEMUA	JSR	CHGBYT	
1016	E56A	09		DEX		
1017	E56B	DF 7F		STX	PTRMEM	
1018	E56D	7E E5 16		JMP	MEM1	output cr, addr, contents
1019	E570	BD E2 FD	MEMSL	JSR	CHGBYT	
1020	E573	7E E5 16		JMP	MEM1	output cr, addr, contents

```

1021 E576 BD E2 FD MEMCR      JSR   CHGBYT
1022 E579 39          RTS           exit task
1023
1024 *****
1025 * move <src1> <src2> [<dest>] - move
1026 *block at <src1> to <src2> to <dest>.
1027 * Moves block 1 byte up if no <dest>.
1028 *****
1029
1030 E57A BD E2 B2 MOVE        JSR   BUFFARG
1031 E57D 7D 00 7E          TST   COUNT
1032 E580 27 2D          BEQ   MOVERR      jump if no arg
1033 E582 BD E3 9B          JSR   WCHEK
1034 E585 26 28          BNE   MOVERR      jump if no delim
1035 E587 DE 83          LDX   SHFTREG      src1
1036 E589 DF 83          STX   PTR1
1037 E58B BD E2 B2          JSR   BUFFARG
1038 E58E 7D 00 7E          TST   COUNT
1039 E591 27 1C          BEQ   MOVERR      jump if no arg
1040 E593 BD E3 A6          JSR   DCHEK
1041 E596 26 17          BNE   MOVERR      jump if no delim
1042 E598 DE 68          LDX   SHFTREG      src2
1043 E59A DF 85          STX   PTR2
1044 E59C BD E2 B2          JSR   BUFFARG
1045 E59F BD E3 8D          JSR   WSKIP
1046 E5A2 81 0D          CMPA  #$0D
1047 E5A4 26 09          BNE   MOVERR      jump if not cr
1048 E5A6 7D 00 7E          TST   COUNT
1049 E5A9 27 0B          BEQ   MOVE1      jump if no arg
1050 E5AB DE 68          LDX   SHFTREG      dest
1051 E5AD 20 0A          BRA   MOVE2
1052 E5AF CE E4 B8 MOVERR      LDX   #MSG9      "bad argument"
1053 E5B2 BD E2 08          JSR   OUTSTRG
1054 E5B5 39          RTS
1055 E5B6 DE 83 MOVE1        LDX   PTR1
1056 E5B8 08          INX           default dest
1057 E5B9 DF 87 MOVE2        STX   PTR3
1058 E5BB DE 87          LDX   PTR3      dest
1059 E5BD 9C 83          CPX   PTR1      src1
1060 E5BF 23 2C          BLS   MOVE3      jump if dest =< src1
1061 E5C1 9C 85          CPX   PTR2      src2
1062 E5C3 22 28          BHI   MOVE3      jump if dest > src2
1063 E5C5 DC 85          LDD   PTR2
1064 E5C7 93 83          SUBD  PTR1
1065 E5C9 D3 87          ADDD  PTR3
1066 E5CB DD 87          STD   PTR3      dest = dest+(src2-src1)
1067 E5CD DE 85          LDX   PTR2
1068 E5CF A6 00 MOVE1P1      LDAA  0,X      char at src2
1069 E5D1 3C          PSHX
1070 E5D2 DE 87          LDX   PTR3
1071 E5D4 8C B6 00          CPX   #$B600      jump if not eeprom

```

1072	E5D7	25 08		BLO	MOVEA	
1073	E5D9	8C B7 FF		CPX	#\$B7FF	jump if not eeprom
1074	E5DC	22 03		BHI	MOVEA	
1075	E5DE	BD E6 0D		JSR	MOVPROG	program eeprom
1076	E5E1	A7 00	MOVEA	STAA	0,X	dest
1077	E5E3	09		DEX		
1078	E5E4	DF 87		STX	PTR3	
1079	E5E6	38		PULX		
1080	E5E7	09		DEX		
1081	E5E8	9C 83		CPX	PTR1	
1082	E5EA	24 E3		BHS	MOVELP1	loop src2-src1 times
1083	E5EC	39		RTS		
1084	E5ED	DE 83	MOVE3	LDX	PTR1	src1
1085	E5EF	A6 00	MOVELP2	LDAA	0,X	
1086	E5F1	3C		PSHX		
1087	E5F2	DE 87		LDX	PTR3	dest
1088	E5F4	8C B6 00		CPX	#\$B600	jump if not eeprom
1089	E5F7	25 08		BLO	MOVEB	
1090	E5F9	8C B7 FF		CPX	#\$B7FF	jump if not eeprom
1091	E5FC	22 03		BHI	MOVEB	
1092	E5FE	BD E6 0D		JSR	MOVPROG	program eeprom
1093	E601	A7 00	MOVEB	STAA	0,X	
1094	E603	08		INX		
1095	E604	DF 87		STX	PTR3	
1096	E606	38		PULX		
1097	E607	08		INX		
1098	E608	9C 85		CPX	PTR2	
1099	E60A	23 E3		BLS	MOVELP2	loop src2-src1 times
1100	E60C	39		RTS		
1101						
1102						
1103						
1104						
1105						
1106						
1107	E60D	37	MOVPROG	PSHB		
1108	E60E	3C		PSHX		
1109	E60F	C6 02		LDAB	#\$02	
1110	E611	F7 10 3B		STAB	PPROG	set eelat
1111	E614	A7 00		STAA	0,X	
1112	E616	C6 03		LDAB	#\$03	
1113	E618	F7 10 3B		STAB	PPROG	set pgm
1114	E61B	CE 08 00		LDX	#\$0800	
1115	E61E	09	MOVEDLY	DEX		
1116	E61F	26 FD		BNE	MOVEDLY	delay 12 ms
1117	E621	C6 00		LDAB	#\$00	
1118	E623	F7 10 3B		STAB	PPROG	
1119	E626	38		PULX		
1120	E627	33		PULB		
1121	E628	39		RTS		
1122						

* MOVPROG - Program eeprom location in X with

* data in A.

1174	E68B	DC 83		LDD	PTR1	
1175	E68D	83 00 08		SUBD	#8	
1176	E690	DD 83		STD	PTR1	
1177	E692	7E E6 C4		JMP	DUMP45	the same code as the SCI.
1178	E695	BD E1 F7	DUMPSCI	JSR	OUTCRLF	
1179	E698	CE 00 83		LDX	#PTR1	
1180	E69B	BD E4 1B		JSR	OUT2BSP	first address
1181	E69E	DE 83		LDX	PTR1	base address
1182	E6A0	5F		CLRB		loop counter
1183	E6A1	BD E4 1E	DUMPDAT	JSR	OUT1BSP	hex value loop
1184	E6A4	5C		INCB		
1185	E6A5	C1 10		CMPB	#\$10	
1186	E6A7	2D F8		BLT	DUMPDAT	loop 16 times
1187	E6A9	5F		CLRB		loop counter
1188	E6AA	DE 83	DUMPASC	LDX	PTR1	base address
1189	E6AC	3A		ABX		
1190	E6AD	A6 00		LDA	0,X	ascii value loop
1191	E6AF	81 20		CMPA	#\$20	
1192	E6B1	25 04		BLO	DUMP3	jump if non printable
1193	E6B3	81 7A		CMPA	#\$7A	
1194	E6B5	23 02		BLS	DUMP4	jump if printable
1195	E6B7	86 20	DUMP3	LDA	#\$20	space for non printables
1196	E6B9	BD E1 76	DUMP4	JSR	OUTPUT	output ascii value
1197	E6BC	5C		INCB		
1198	E6BD	C1 10		CMPB	#\$10	
1199	E6BF	2D E9		BLT	DUMPASC	loop 16 times
1200	E6C1	BD E3 B2		JSR	CHKABRT	check abort or wait
1201	E6C4	DC 83	DUMP45	LDD	PTR1	
1202	E6C6	C3 00 10		ADDD	#\$10	point to next 16 byte bound
1203	E6C9	DD 83		STD	PTR1	update ptr1
1204	E6CB	1A 93 85		CPD	PTR2	
1205	E6CE	22 0E		BHI	DUMP5	quit if ptr1 > ptr2
1206	E6D0	1A 83 00 00		CPD	#\$00	check wraparound at \$ffff
1207	E6D4	26 A4		BNE	DUMPLP	jump - no wraparound
1208	E6D6	DC 85		LDD	PTR2	
1209	E6D8	1A 83 FF F0		CPD	#\$FFF0	
1210	E6DC	25 9C		BLO	DUMPLP	upper bound not at top
1211	E6DE	39	DUMP5	RTS		quit
1212						
1213						*****
1214						* screen() - this subroutine outputs 8 bytes of data in sequence
1215						* starting at the address in PTR1. Data is formatted for
1216						* the CRT screen.
1217						*****
1218						
1219	E6DF	BD E2 4E	SCREEN	JSR	CLRBUFF	clear display buffer
1220	E6E2	CE 00 83		LDX	#PTR1	
1221	E6E5	BD E4 1B		JSR	OUT2BSP	output address
1222	E6E8	BD E4 21		JSR	OUTSPAC	for screen format
1223	E6EB	DE 83		LDX	PTR1	base address
1224	E6ED	5F		CLRB		loop counter

1276	E749	08		INX	
1277	E74A	81 53		CMPA #'S'	s is last register
1278	E74C	26 F8		BNE RPR11	jump if not done
1279	E74E	7D 00 01		TST IODEV	
1280	E751	27 06		BEQ RPR2	
1281	E753	BD E1 37		JSR INCHAR	display data and wait
1282	E756	BD E2 4E		JSR CLRBUFF	
1283	E759	38	RPR2	PULX	
1284	E75A	39		RTS	
1285					
1286				*****	
1287				* register [<name>] - prints the user regs	
1288				*and opens them for modification. <name> is	
1289				*the first register opened (default = P).	
1290				* Subcommands:	
1291				* [<nn>]<cr> Opens the next register.	
1292				* [<nn>]<. > Return.	
1293				* The register value is only changed if	
1294				* <nn> is entered before the subcommand.	
1295				*****	
1296					
1297	E75B	CE E6 FA	REGISTER	LDX #REGLIST	
1298	E75E	BD E3 8D		JSR WSKIP	a = first char of arg
1299	E761	BD E2 74		JSR UPCASE	convert to upper case
1300	E764	81 0D		CMPA #\$D	
1301	E766	27 1E		BEQ REG4	jump if no argument
1302	E768	A1 00	REG1	CMPA	0,X
1303	E76A	27 0E		BEQ REG3	
1304	E76C	E6 00		LDAB 0,X	
1305	E76E	08		INX	
1306	E76F	C1 53		CMPB #'S'	
1307	E771	26 F5		BNE REG1	jump if not "s"
1308	E773	CE E4 B8	REG2	LDX #MSG9	"bad argument"
1309	E776	BD E2 08		JSR OUTSTRG	
1310	E779	39		RTS	
1311	E77A	3C	REG3	PSHX	
1312	E77B	BD E3 7F		JSR INCBUFF	
1313	E77E	BD E3 8D		JSR WSKIP	next char after arg
1314	E781	81 0D		CMPA #\$D	
1315	E783	38		PULX	
1316	E784	26 ED		BNE REG2	jump if not cr
1317	E786	BD E7 3F	REG4	JSR RPRINT	print all registers
1318	E789	BD E1 F7	REG5	JSR OUTCRLF	
1319	E78C	BD E2 4E	REG51	JSR CLRBUFF	
1320	E78F	BD E7 0F		JSR RPRNT1	print reg name
1321	E792	86 3E		LDAA #PROMPT	
1322	E794	BD E1 76		JSR OUTPUT	output prompt character
1323	E797	BD E2 D5		JSR TERMARG	read subcommand
1324	E79A	BD E3 A6		JSR DCHEK	
1325	E79D	27 07		BEQ REG6	jump if delimiter
1326	E79F	CE E4 B8		LDX #MSG9	"bad argument"

1327	E7A2	BD E2 08		JSR	OUTSTRG	
1328	E7A5	39		RTS		
1329	E7A6	36	REG6	PSHA		
1330	E7A7	3C		PSHX		
1331	E7A8	7D 00 7E		TST	COUNT	
1332	E7AB	27 14		BEQ	REG8	jump if no input
1333	E7AD	E6 07		LDAB	7,X	get reg offset
1334	E7AF	A6 0E		LDAA	14,X	byte size
1335	E7B1	CE 00 72		LDX	#REGS	user registers
1336	E7B4	3A		ABX		
1337	E7B5	4D		TSTA		
1338	E7B6	27 05		BEQ	REG7	jump if 1 byte reg
1339	E7B8	96 68		LDAA	SHFTREG	
1340	E7BA	A7 00		STAA	0,X	put in top byte
1341	E7BC	08		INX		
1342	E7BD	96 69	REG7	LDAA	SHFTREG+1	
1343	E7BF	A7 00		STAA	0,X	put in bottom byte
1344	E7C1	38	REG8	PULX		
1345	E7C2	32		PULA		
1346	E7C3	E6 00		LDAB	0,X	CHECK FOR REGISTER S
1347	E7C5	C1 53		CMPB	#S'	
1348	E7C7	27 05		BEQ	REG9	jump if "s"
1349	E7C9	08		INX		point to next register
1350	E7CA	81 2E		CMPA		#.'
1351	E7CC	26 BB		BNE	REG5	jump if not period
1352	E7CE	39	REG9	RTS		
1353						
1354						
1355						
1356						
1357	E7CF	CE E7 D6	HELP	LDX	#HELPMMSG1	
1358	E7D2	BD E2 08		JSR	OUTSTRG	print screen
1359	E7D5	39		RTS		
1360						
1361	E7D6	2A 20 4D 65HELPMMSG1		FCC		'* Memory commands *'
1362	E7E9	0D		FCB	#\$0D	
1363	E7EA	4D 44 20 28		FCC	'MD	(<addr1> (<addr2>)) memory dump.
1364					MV	<s1> <s2> (<d>) block move.'
1365	E82F	0D		FCB	#\$0D	
1366	E830	4D 4D 20 28		FCC	'MM	(<addr>) Open memory location. Subcommands:'
1367						
1368	E85F	0D		FCB	#\$0D	
1369	E860	3C 3D 3E 20		FCC		'<=> open same address. <-> open previous address.'
1370						
1371	E894	0D		FCB	#\$0D	
1372	E895	3C 63 72 3E		FCC		'<cr> open next address. <sp> open next address.'
1373						

1374	E8C7	0D	FCB	#\$0D	
1375	E8C8	20 20 20 20	FCC		' <.> quit.'
1376	E8D6	0D	FCB	#\$0D	
1377	E8D7	41 53 4D 20	FCC		'ASM (<addr>) line assembler/disassembler. Subcommands:'
1378					
1379	E90D	0D	FCB	#\$0D	
1380	E90E	3C 3D 3E 20	FCC		'<=> do same address. <-> do previous address.'
1381					
1382	E93D	0D	FCB	#\$0D	
1383	E93E	3C 63 72 3E	FCC		'<cr> do next opcode. <.> quit.'
1384	E95F	0D	FCB	#\$0D	
1385	E960	2A 20 43 6F	FCC		'* Code execution commands *'
1386	E97B	0D	FCB	#\$0D	
1387	E97C	42 52 4B 20	FCC	'BRK	'(<.>(<addr>) Set up breakpoint table.'
1388	E9A5	0D	FCB	#\$0D	
1389	E9A6	52 44 20 28	FCC	'RD	'(p,y,x,a,b,c,s) Open user registers.'
1390	E9CE	0D	FCB	#\$0D	
1391	E9CF	47 20 28 3C	FCC	'G	'(<addr>) execute user code with breakpoints.'
1392					
1393	E9FE	0D	FCB	#\$0D	
1394	E9FF	45 58 20 3C	FCC	'EX	'(<addr>) execute user code (no breakpoints).'
1395					
1396	EA2E	0D	FCB	#\$0D	
1397	EA2F	43 41 4C 4C	FCC	'CALL	'(<addr>) Call user subroutine.'
1398	EA53	0D	FCB	#\$0D	
1399	EA54	54 20 28 3C	FCC	'T	'(<n>) Trace n instructions. continue executing.'
1400				P	
1401	EA8B	0D	FCB	#\$0D	
1402	EA8C	2A 20 4F 74	FCC		'* Other useful commands *'
1403	EA 15	0D	FCB	#\$0D	
1404	EAA6	42 55 4C 4B	FCC	'BULK	'Erase the eeprom. BULKALL Erase eeprom and config.'
1405					
1406	EAE5	0D	FCB	#\$0D	
1407	EAE6	43 4E 54 4C	FCC		'CNTLX or DEL Abort.'
1408	EAFB	0D	FCB	#\$0D	
1409	EAPC	04	FCB	4	
1410					
1411					*****
1412					* bulk - Bulk erase the eeprom except the
1413					*config register.
1414					*****
1415					
1416	EAFD	7F 00 92	BULK	CLR	TMP2
1417	EB00	20 06		BRA	BULK1
1418					
1419					*****
1420					* bulkall - Bulk erase the eeprom and the

```

1421                                     *config register.
1422                                     *****
1423
1424 EB02 7F 00 92 BULKALL CLR TMP2
1425 EB05 7C 00 92 INC TMP2
1426 EB08 36 BULK1 PSHA
1427 EB09 86 06 LDAA #S06
1428 EB0B B7 10 3B STAA PPROG set eelat, erase bits
1429 EB0E 86 FF LDAA #SFF
1430 EB10 7D 00 92 TST TMP2
1431 EB13 26 05 BNE BULK2 jump if config
1432 EB15 B7 B6 00 STAA $B600 write to $b600
1433 EB18 20 03 BRA BULK3
1434 EB1A B7 10 3F BULK2 STAA CONFIG
1435 EB1D 86 07 BULK3 LDAA #S07
1436 EB1F B7 10 3B STAA PPROG
1437 EB22 3C PSHX
1438 EB23 CE 0F A0 LDX #4000 4000*5~
1439 EB26 09 BULKDLY DEX 2~
1440 EB27 26 FD BNE BULKDLY 3~
1441 EB29 38 PULX
1442 EB2A 7F 10 3B CLR PPROG
1443 EB2D 32 PULA
1444 EB2E 39 RTS
1445
1446                                     *****
1447                                     * call [<addr>] - Execute a jsr to addr or
1448                                     *user's pc value. Return to monitor by rts
1449                                     *or breakpoint.
1450                                     *****
1451
1452 EB2F BD E3 8D CALL JSR WSKIP
1453 EB32 81 0D CMPA #SD
1454 EB34 27 15 BEQ CALL3 jump if no arg
1455 EB36 BD E2 B2 JSR BUFFARG
1456 EB39 BD E3 8D JSR WSKIP
1457 EB3C 81 0D CMPA #SD
1458 EB3E 27 07 BEQ CALL2 jump if cr
1459 EB40 CE E4 B8 LDX #MSG9 "bad argument"
1460 EB43 BD E2 08 JSR OUTSTRG
1461 EB46 39 RTS
1462 EB47 DE 68 CALL2 LDX SHFTREG
1463 EB49 DF 72 STX REGS pc = <addr>
1464 EB4B DE 7B CALL3 LDX SP
1465 EB4D 09 DEX user stack pointer
1466 EB4E CC EB 5D LDD #RETURN return address
1467 EB51 ED 00 STD 0,X
1468 EB53 09 DEX
1469 EB54 DF 7B STX SP new user stack pointer
1470 EB56 C6 02 LDAB #2 (2=CALL)
1471 EB58 D7 92 STAB TMP2 flag for breakpoints

```

```

1472 EB5A 7E EB D4          JMP  TRACE3      pickup trace command
1473
1474          *****
1475          * return() - Return here from rts after
1476          * call command.
1477          *****
1478
1479 EB5D BD E1 F7  RETURN    JSR  OUTCRLF
1480 EB60 BD E7 3F          JSR  RPRINT      print user registers
1481 EB63 7E E0 3E          JMP  MAIN
1482
1483          *****
1484          * go [<addr>] - Execute starting at <addr> or
1485          * user's pc value. Executes an rti to user code.
1486          * Returns to monitor via a breakpoint.
1487          *****
1488
1489 EB66 BD E3 8D  GO        JSR  WSKIP
1490 EB69 81 0D          CMPA  #$0D
1491 EB6B 27 15          BEQ   GO2          jump if no arg
1492 EB6D BD E2 B2          JSR  BUFFARG
1493 EB70 BD E3 8D          JSR  WSKIP
1494 EB73 81 0D          CMPA  #$0D
1495 EB75 27 07          BEQ   GO1          jump if cr
1496 EB77 CE E4 B8          LDX  #MSG9      "bad argument"
1497 EB7A BD E2 08          JSR  OUTSTRG
1498 EB7D 39            RTS
1499 EB7E DE 68  GO1        LDX  SHFTREG
1500 EB80 DF 72          STX   REGS      pc = <addr>
1501 EB82 C6 03  GO2        LDAB #3      (0=T,1=P,2=CALL,3=G)
1502 EB84 D7 92          STAB  TMP2
1503 EB86 7E EB D4          JMP  TRACE3      do trace
1504
1505          *****
1506          * execute [<addr>] - executes
1507          * user program starting at <addr>
1508          * or user's pc if no <addr> supplied.
1509          * No breakpoints are inserted.
1510          *****
1511
1512 EB89 BD E3 8D  EX        JSR  WSKIP
1513 EB8C 81 0D          CMPA  #$0D
1514 EB8E 27 15          BEQ   EX2          jump if no arg
1515 EB90 BD E2 B2          JSR  BUFFARG
1516 EB93 BD E3 8D          JSR  WSKIP
1517 EB96 81 0D          CMPA  #$0D
1518 EB98 27 07          BEQ   EX1          jump if cr
1519 EB9A CE E4 B8          LDX  #MSG9      "bad argument"
1520 EB9D BD E2 08          JSR  OUTSTRG
1521 EBA0 39            RTS
1522 EBA1 DE 68  EX1        LDX  SHFTREG

```

```

1523 EBA3 DF 72          STX  REGS          pc = <addr>
1524 EBA5 7E EC 4E      EX2          JMP  RESTACK
1525
1526 *****
1527 * proceed - Same as go except it ignores
1528 * a breakpoint at the first opcode. Calls
1529 * trace once and then go.
1530 *****
1531
1532 EBA8 C6 01          PROCEED      LDAB  #1          (0=T,1=P,2=CALL,3=G)
1533 EBAA D7 92          STAB  TMP2
1534 EBAC 7E EB D4          JMP  TRACE3
1535
1536 *****
1537 * trace <n> - Trace n instructions starting
1538 * at user's pc value. n is a hex number less than
1539 * $FF (defaults to 1).
1540 *****
1541
1542 EBAF 7F 00 94      TRACE          CLR  TMP4
1543 EBB2 7C 00 94      INC  TMP4          default count1 = 1
1544 EBB5 5F          CLR  B          (0=T,1=P,2=CALL,3=G)
1545 EBB6 D7 92          STAB  TMP2
1546 EBB8 BD E3 8D      JSR  WSKIP
1547 EBBB 81 0D      CMPA  #$0D
1548 EBBD 27 15      BEQ  TRACE3          jump if cr
1549 EBBF BD E2 B2      JSR  BUFFARG
1550 EBC2 BD E3 8D      JSR  WSKIP
1551 EBC5 81 0D      CMPA  #$0D
1552 EBC7 27 07      BEQ  TRACE1          jump if cr
1553 EBC9 CE E4 B8      LDX  #MSG9          "bad argument"
1554 EBCC BD E2 08      JSR  OUTSTRG
1555 EBCF 39          RTS
1556 EBD0 96 69      TRACE1          LDAA  SHFTREG+1      n
1557 EBD2 97 94          STAA  TMP4
1558 EBD4 B6 10 20      TRACE3          LDAA  TCTL1
1559 EBD7 97 85          STAA  PTR2          save user mode/level
1560 EBD9 84 FC          ANDA  #SFC
1561 EBD8 B7 10 20      STAA  TCTL1          disable oc5 output
1562 EBDE B6 10 22      LDAA  TMSK1
1563 EBE1 97 86          STAA  PTR2+1          save user int masks
1564 EBE3 7F 10 24      CLR  TMSK2          disable tof and pac ints
1565 EBE6 DE 9C          LDX  VOC5+1
1566 EBE8 DF 89          STX  PTR4          save user's vector
1567 EBEA 86 7E          LDAA  #$7E          jmp opcode
1568 EBEC 97 9B          STAA  VOC5
1569 EBEE CE EC 0E      LDX  #TRACEIN
1570 EBF1 DF 9C          STX  VOC5+1          monitor oc5 vector
1571 EBF3 96 7A          LDAA  REGS+8          user ccr
1572 EBF5 84 EF          ANDA  #SEF          clear i bit
1573 EBF7 97 7A          STAA  REGS+8

```

1574	EBF9	C6 57		LDAB #87	cycles to end of rti
1575	EBFB	FE 10 0E		LDX TCNT	timer count value
1576	EBFE	3A		ABX	3~ \
1577	EBFF	FF 10 1E		STX TOC5	oc5 match register 5~ \
1578	EC02	86 08		LDAA #508	2~ \
1579	EC04	B7 10 23		STAA TFLG1	clear oc5 int flag 4~ \
1580	EC07	B7 10 22		STAA TMSK1	enable oc5 int 4~ / 86~
1581	EC0A	0E		CLI	2~ /
1582	EC0B	7E EC 4E		JMP RESTACK	execute an rti 66~ /
1583					
1584				*****	
1585				* tracein - return from toc5 interrupt.	
1586				*****	
1587					
1588	EC0E	0F	TRACEIN	SEI	
1589	EC0F	7F 10 22		CLR TMSK1	disable timer ints
1590	EC12	30		TSX	
1591	EC13	8E 00 5F		LDS #STACK	
1592	EC16	BD EC 67		JSR SAVSTACK	save user regs
1593	EC19	DE 89		LDX PTR4	
1594	EC1B	DF 9C		STX VOC5+1	
1595	EC1D	7D 00 01		TST IODEV	
1596	EC20	26 03		BNE TRACEIN1	
1597	EC22	BD E3 B2		JSR CHKABRT	check for abort
1598	EC25	D6 92	TRACEIN1	LDAB TMP2	
1599	EC27	27 17		BEQ TRACE9	jump if trace command
1600	EC29	C1 01		CMPB #1	proceed command?
1601	EC2B	26 03		BNE TRACEIN2	no.
1602	EC2D	7E EB 82		JMP GO2	yes-pickup go command
1603	EC30	BD EC 89	TRACEIN2	JSR BRPTCHK	check for breakpt (call or go)
1604	EC33	DE 72		LDX REGS	get user pc
1605	EC35	8C EB 5D		CPX #RETURN	see if we reached RTS
1606	EC38	27 03		BEQ TRACEIN3	yes - return control
1607	EC3A	7E EB D4		JMP TRACE3	no - trace another instruction
1608	EC3D	7E EB 5D	TRACEIN3	JMP RETURN	
1609	EC40	BD E1 F7	TRACE9	JSR OUTCRLF	print registers for
1610	EC43	BD E7 3F		JSR RPRINT	trace only.
1611	EC46	7A 00 94		DEC TMP4	
1612	EC49	22 89		BHI TRACE3	jump if count1 >= 0
1613	EC4B	7E E0 3E		JMP MAIN	return to monitor
1614					
1615				*****	
1616				* restack() - Restore user stack and	
1617				*execute an RTI. Extended addressing forced	
1618				*to ensure count value for trace.	
1619				*****	
1620					
1621	EC4E	BE 00 7B	RESTACK	LDS >SP	stack pointer
1622	EC51	FE 00 72		LDX >REGS	
1623	EC54	3C		PSHX	pc
1624	EC55	FE 00 74		LDX >REGS+2	

```

1625 EC58 3C          PSHX          y
1626 EC59 FE 00 76   LDX    >REGS+4
1627 EC5C 3C          PSHX          x
1628 EC5D FC 00 78   LDD    >REGS+6
1629 EC60 36          PSHA          a
1630 EC61 37          PSHB          b
1631 EC62 B6 00 7A   LDAA   >REGS+8
1632 EC65 36          PSHA          ccr
1633 EC66 3B          RESTACK1 RTI
1634
1635 *****
1636 * savstack() - Save user's registers. If originating command
1637 * is CALL, then don't update the user's pc.
1638 *****
1639
1640 EC67 A6 00   SAVSTACK LDAA  0,X
1641 EC69 97 7A   STAA  REGS+8      ccr
1642 EC6B EC 01   LDD    1,X
1643 EC6D 97 79   STAA  REGS+7      b
1644 EC6F D7 78   STAB  REGS+6      a
1645 EC71 EC 03   LDD    3,X
1646 EC73 DD 76   STD   REGS+4      x
1647 EC75 EC 05   LDD    5,X
1648 EC77 DD 74   STD   REGS+2      y
1649 EC79 96 92   LDAA  TMP2      check to see if call command
1650 EC7B 81 02   CMPA  #2
1651 EC7D 27 04   BEQ   SAVST1      yes- then don't change pc
1652 EC7F EC 07   LDD    7,X
1653 EC81 DD 72   STD   REGS      pc
1654 EC83 C6 08   SAVST1 LDAB  #8
1655 EC85 3A      ABX
1656 EC86 DF 7B   STX   SP          stack pointer
1657 EC88 39      RTS
1658
1659 *****
1660 * BRPTCHK() - checks to see if
1661 * a breakpoint has been reached
1662 *****
1663
1664 EC89 3C          BRPTCHK  PSHX          save (X)
1665 EC8A 5F          CLR      CLRB         offset into breakpoint table
1666 EC8B CE 00 6A   BRPTCHK1 LDX    #BRKTABL point to table
1667 EC8E 3A          ABX          add offset
1668 EC8F EE 00      LDX    0,X          get breakpoint
1669 EC91 9C 72      CPX    REGS        see if it matches user PC
1670 EC93 27 08      BEQ   BRPTCHK2
1671 EC95 CB 02      ADDB  #2          point to next entry in table
1672 EC97 C1 06      CMPB  #6          done with search?
1673 EC99 2F F0      BLE   BRPTCHK1      no-check next entry
1674 EC9B 38          PULX          restore (X)
1675 EC9C 39          RTS          no match found-return

```



```

1676 EC9D D6 92      BRPTCHK2    LDAB TMP2
1677 EC9F C1 02      CMPB #2          call command?
1678 ECA1 26 06      BNE BPHIT1       no
1679 ECA3 DE 7B      LDX SP          remove return address
1680 ECA5 08          INX
1681 ECA6 08          INX
1682 ECA7 DF 7B      STX SP
1683 ECA9 BD E1 F7    BPHIT1    JSR OUTCRLF
1684 ECAC BD E7 3F    JSR RPRINT    display registers
1685 ECAF 7E E0 3E    JMP MAIN      return to monitor
1686
1687 *****
1688 * break [-][<addr>]...
1689 * Modifies the breakpoint table. More than
1690 * one argument can be entered on the command
1691 * line but the table will hold only 4 entries.
1692 * 4 types of arguments are implied above:
1693 * break      Prints table contents.
1694 * break <addr> Inserts <addr>.
1695 * break -<addr> Deletes <addr>.
1696 * break -    Clears all entries.
1697 *****
1698
1699 ECB2 BD E3 8D    BREAK      JSR WSKIP
1700 ECB5 81 0D      CMPA #$0D
1701 ECB7 26 04      BNE BRKDEL    jump if not cr
1702 ECB9 BD ED 2D    JSR BPRINT    print table
1703 ECBC 39          RTS
1704 ECBD 81 2D      BRKDEL      CMPA #'- '
1705 ECBF 26 31      BNE BRKDEF    jump if not -
1706 ECC1 BD E3 7F    JSR INCBUFF
1707 ECC4 BD E3 78    JSR RDBUFF
1708 ECC7 BD E3 A6    JSR DCHEK
1709 ECCA 26 06      BNE BRKDEL1    jump if not delimiter
1710 ECCC BD E3 6C    JSR BPCLR      clear table
1711 ECCF 7E EC B2    JMP BREAK      do next argument
1712 ECD2 BD E2 B2    BRKDEL1    JSR BUFFARG    get address to delete
1713 ECD5 BD E3 A6    JSR DCHEK
1714 ECD8 27 07      BEQ BRKDEL2    jump if delimiter
1715 ECDA CE E4 B8    LDX #MSG9      "bad argument"
1716 ECDD BD E2 08    JSR OUTSTRG
1717 ECE0 39          RTS
1718 ECE1 BD ED 4C    BRKDEL2    JSR BPSRCH      look for addr in table
1719 ECE4 5D          TSTB
1720 ECE5 2B 08      BMI BRKDEL3    jump if not found
1721 ECE7 CE 00 6A    LDX #BRKTABL
1722 ECEA 3A          ABX
1723 ECEB 6F 00      CLR 0,X        clear entry
1724 ECED 6F 01      CLR 1,X
1725 ECEF 7E EC B2    BRKDEL3    JMP BREAK      do next argument
1726 ECF2 BD E2 B2    BRKDEF      JSR BUFFARG    get argument

```

```

1727 ECF5 BD E3 A6      JSR  DCHEK
1728 ECF8 27 07        BEQ  BRKDEF1      jump if delimiter
1729 ECFA CE E4 B8      LDX  #MSG9        "bad argument"
1730 ECFD BD E2 08      JSR  OUTSTRG
1731 ED00 39            RTS
1732 ED01 BD ED 4C      BRKDEF1 JSR  BPSRCH      look for entry in table
1733 ED04 5D            TSTB
1734 ED05 2C AB        BGE  BREAK      jump if already in table
1735 ED07 DE 68        LDX  SHFTREG
1736 ED09 7F 00 68      CLR  SHFTREG
1737 ED0C 7F 00 69      CLR  SHFTREG+1
1738 ED0F 3C            PSHX
1739 ED10 BD ED 4C      JSR  BPSRCH      look for 0 entry
1740 ED13 38            PULX
1741 ED14 5D            TSTB
1742 ED15 2A 0A        BPL  BRKDEF3      jump if table not full
1743 ED17 CE E4 A0      LDX  #MSG4        "full"
1744 ED1A BD E2 08      JSR  OUTSTRG
1745 ED1D BD ED 2D      JSR  BPRINT
1746 ED20 39            RTS
1747 ED21 18 CE 00 6A   BRKDEF3 LDY  #BRKTABL
1748 ED25 18 3A        ABY
1749 ED27 CD EF 00      STX  0,Y          put new entry in
1750 ED2A 7E EC B2      JMP  BREAK        do next argument
1751
1752 *****
1753 * bprint() - print the contents of the table.
1754 *****
1755
1756 ED2D BD E1 F7      BPRINT JSR  OUTCRLF
1757 ED30 BD E2 4E      JSR  CLRBUFF      clear display buffer
1758 ED33 CE 00 6A      LDX  #BRKTABL
1759 ED36 C6 04        LDAB #4
1760 ED38 BD E4 1B      BPRINT1 JSR  OUT2BSP
1761 ED3B BD E4 21      JSR  OUTSPAC      for formatting
1762 ED3E 5A            DECB
1763 ED3F 2E F7        BGT  BPRINT1      loop 4 times
1764 ED41 D6 01        LDAB IODEV
1765 ED43 27 06        BEQ  BPRINT2
1766 ED45 BD F9 EB      JSR  VIDEO        display until user hits key
1767 ED48 BD E2 4E      JSR  CLRBUFF
1768 ED4B 39            BPRINT2 RTS
1769
1770 *****
1771 * bpsrch() - search table for address in
1772 * shftreg. Returns b = index to entry or
1773 * b = -1 if not found.
1774 *****
1775
1776 ED4C 5F            BPSRCH  CLRB
1777 ED4D CE 00 6A      BPSRCH1 LDX  #BRKTABL

```

1778	ED50	3A		ABX		
1779	ED51	EE 00		LDX	0,X	get table entry
1780	ED53	9C 68		CPX	SHFTREG	
1781	ED55	26 01		BNE	BPSRCH2	jump if no match
1782	ED57	39		RTS		
1783	ED58	5C	BPSRCH2	INCB		
1784	ED59	5C		INCB		
1785	ED5A	C1 06		CMPB	#\$6	
1786	ED5C	2F EF		BLE	BPSRCH1	loop 4 times
1787	ED5E	C6 FF		LDAB	#\$FF	
1788	ED60	39		RTS		
1789				*****		
1790				* load(ptrbuff[]) - Load s1/s9 records from		
1791				*host to memory. Ptrbuff[] points to string in		
1792				*input buffer which is a command to output s1/s9		
1793				*records from the host ("cat filename" for unix).		
1794				* Returns error and address if it can't write		
1795				*to a particular location.		
1796				*****		
1797				* verify(ptrbuff[]) - Verify memory from load		
1798				*command. Ptrbuff[] is same as for load.		
1799				*****		
1800						
1801	ED61	7F 00 92	VERIFY	CLR	TMP2	
1802	ED64	7C 00 92		INC	TMP2	flagt1 = 1 = verify
1803	ED67	20 03		BRA	LOAD1	
1804	ED69	7F 00 92	LOAD	CLR	TMP2	flagt1 = 0 = load
1805						
1806	ED6C	7D 00 01	LOAD1	TST	IODEV	
1807	ED6F	27 07		BEQ	LOAD1A	
1808	ED71	CE E4 E5		LDX	#\$MSG14	'NOT USING SCI'
1809	ED74	BD E2 08		JSR	OUTSTRG	
1810	ED77	39		RTS		
1811	ED78	BD E3 8D	LOAD1A	JSR	WSKIP	
1812	ED7B	81 0D		CMPA	#\$0D	
1813	ED7D	27 07		BEQ	LOAD5A	
1814	ED7F	CE E4 B8		LDX	#\$MSG9	bad argument
1815	ED82	BD E2 08		JSR	OUTSTRG	
1816	ED85	39		RTS		
1817	ED86	7F 00 7D	LOAD5A	CLR	AUTOLF	
1818	ED89	BD E3 B2	LOAD5	JSR	CHKABRT	
1819	ED8C	81 02		CMPA	#\$CTLA	
1820	ED8E	26 04		BNE	LOAD6	
1821	ED90	7C 00 7D		INC		AUTOLF
1822	ED93	39		RTS		
1823	ED94	BD E1 41	LOAD6	JSR	INPUT	read SCI
1824	ED97	4D		TSTA		
1825	ED98	27 EF		BEQ	LOAD5	jump if no input
1826	ED9A	81 53		CMPA	#\$S'	
1827	ED9C	26 EB		BNE	LOAD5	jump if not S
1828	ED9E	BD E1 41	LOAD7	JSR	INPUT	

1829	EDA1	4D		TSTA	
1830	EDA2	27 FA		BEQ	LOAD7 jump if no input
1831	EDA4	81 39		CMPA	#9
1832	EDA6	26 0F		BNE	LOAD8 jump if not S9
1833	EDA8	BD EE 0D		JSR	BYTE
1834	EDAB	D6 69		LDAB	SHFTREG+1 b = byte count
1835	EDAD	BD EE 0D	LOAD75	JSR	BYTE
1836	EDB0	5A		DECB	
1837	EDB1	26 FA		BNE	LOAD75 loop until end of record
1838	EDB3	7C 00 7D		INC	AUTOLF turn on autolf
1839	EDB6	39		RTS	all done return normally
1840	EDB7	81 31	LOAD8	CMPA	#'1'
1841	EDB9	26 CE		BNE	LOAD5 jump if not S1
1842	EDBB	7F 00 94		CLR	TMP4 clear checksum
1843	EDBE	BD EE 0D		JSR	BYTE
1844	EDC1	D6 69		LDAB	SHFTREG+1
1845	EDC3	C0 02		SUBB	#S2 b = byte count
1846	EDC5	BD EE 0D		JSR	BYTE
1847	EDC8	BD EE 0D		JSR	BYTE
1848	EDCB	DE 68		LDX	SHFTREG x = base address
1849	EDCD	09		DEX	
1850	EDCE	BD EE 0D	LOAD10	JSR	BYTE get next byte
1851	EDD1	08		INX	
1852	EDD2	5A		DECB	
1853	EDD3	27 23		BEQ	LOAD12 check byte count
1854	EDD5	96 69		LDAA	SHFTREG+1 if 0, go do checksum
1855	EDD7	7D 00 92		TST	TMP2
1856	EDDA	26 03		BNE	LOAD11 jump if verify
1857	EDDC	BD E3 04		JSR	CHGBYT0 put byte into memory
1858	EDDF	A1 00	LOAD11	CMPA	0,X verify ram location
1859	EDE1	27 EB		BEQ	LOAD10 jump if ram ok
1860	EDE3	DF 87		STX	PTR3 save error address
1861	EDE5	7C 00 7D		INC	AUTOLF turn on autolf
1862	EDE8	BD E1 F7		JSR	OUTCRLF
1863	EDEB	CE E4 D9		LDX	#MSG13 "error addr"
1864	EDEE	BD E2 08		JSR	OUTSTRG
1865	EDF1	CE 00 87		LDX	#PTR3
1866	EDF4	BD E4 1B		JSR	OUT2BSP address
1867	EDF7	39		RTS	
1868	EDF8	96 94	LOAD12	LDAA	TMP4
1869	EDFA	4C		INCA	do checksum
1870	EDFB	26 03		BNE	LOAD13 jump if s1 record okay
1871	EDFD	7E ED 89		JMP	LOAD5
1872	EE00	7C 00 7D	LOAD13	INC	AUTOLF
1873	EE03	BD E1 F7		JSR	OUTCRLF
1874	EE06	CE E4 CA		LDX	#MSG12 "checksum error"
1875	EE09	BD E2 08		JSR	OUTSTRG
1876	EE0C	39		RTS	
1877					
1878					
1879					

* byte() - Read 2 ascii bytes and

```

1880                                     *convert to one hex byte. Returns byte
1881                                     *shifted into shftreg and added to tmp4.
1882                                     *****
1883
1884 EE0D 37      BYTE      PSHB
1885 EE0E 3C      PSHX
1886 EE0F BD E1 41  BYTE0    JSR  INPUT      (1st byte)
1887 EE12 4D      TSTA
1888 EE13 27 FA      BEQ  BYTE0      loop until input
1889 EE15 BD E2 7F    JSR  HEXBIN
1890 EE18 BD E1 41  BYTE1    JSR  INPUT      read host (2nd byte)
1891 EE1B 4D      TSTA
1892 EE1C 27 FA      BEQ  BYTE1      loop until input
1893 EE1E BD E2 7F    JSR  HEXBIN
1894 EE21 96 69      LDAA  SHFTREG+1
1895 EE23 9B 94      ADDA  TMP4
1896 EE25 97 94      STAA  TMP4      add to checksum
1897 EE27 38      PULX
1898 EE28 33      PULB
1899 EE29 39      RTS
1900
1901 0000      PAGE1      EQU  $00      values for page opcodes
1902 0018      PAGE2      EQU  $18
1903 001A      PAGE3      EQU  $1A
1904 00CD      PAGE4      EQU  $CD
1905 0000      IMMED      EQU  $0      addressing modes
1906 0001      INDX       EQU  $1
1907 0002      INDY       EQU  $2
1908 0003      LIMMED     EQU  $3      (long immediate)
1909 0004      OTHER      EQU  $4
1910
1911      *** Rename variables for assem/disassem ***
1912 0092      AMODE       EQU  TMP2      addressing mode
1913 0093      YFLAG       EQU  TMP3
1914 0094      PNORM       EQU  TMP4      page for normal opcode
1915 007F      OLDPC       EQU  PTRMEM
1916 0083      PC         EQU  PTR1      program counter
1917 0085      PX         EQU  PTR2      page for x indexed
1918 0086      PY         EQU  PTR2+1    page for y indexed
1919 0087      BASEOP      EQU  PTR3      base opcode
1920 0088      CLASS      EQU  PTR3+1    class
1921 0089      DISPC       EQU  PTR4      pc for disassembler
1922 008B      BRADDR      EQU  PTR5      relative branch offset
1923 008D      MNEPTR      EQU  PTR6      pointer to table for dis
1924 008F      ASSCOMM     EQU  PTR7      subcommand for assembler
1925
1926      *** Error messages for assembler ***
1927 EE2A EE 3C      MSGDIR  FDB  #MSG A1      message table index
1928 EE2C EE 53      FDB  #MSG A2
1929 EE2E EE 6B      FDB  #MSG A3
1930 EE30 EE 7A      FDB  #MSG A4

```

1931	EE32	EE 87		FDB	#MSGAS	
1932	EE34	EE 9A		FDB	#MSGAS6	
1933	EE36	EE B2		FDB	#MSGAS7	
1934	EE38	EE CD		FDB	#MSGAS8	
1935	EE3A	EE DA		FDB	#MSGAS9	
1936	EE3C	49 6D 6D 65MSGAS1		FCC	'Immediate mode illegal'	
1937	EE52	04		FCB	EOT	
1938	EE53	45 72 72 6F MSGAS2		FCC	'Error in mnemonic table'	
1939	EE6A	04		FCB	EOT	
1940	EE6B	49 6C 6C 65MSGAS3		FCC	'Illegal bit op'	
1941	EE79	04		FCB	EOT	
1942	EE7A	42 61 64 20 MSGAS4		FCC	'Bad argument'	
1943	EE86	04		FCB	EOT	
1944	EE87	4D 6E 65 6DMSGAS5		FCC	'Mnemonic not found'	
1945	EE99	04		FCB	EOT	
1946	EE9A	55 6E 6B 6EMSGAS6		FCC	'Unknown addressing mode'	
1947	EEB1	04		FCB	EOT	
1948	EEB2	49 6E 64 65 MSGAS7		FCC	'Indexed addressing assumed'	
1949	EECC	04		FCB	EOT	
1950	EECD	53 79 6E 74 MSGAS8		FCC	'Syntax error'	
1951	EED9	04		FCB	EOT	
1952	EEDA	42 72 61 6E MSGAS9		FCC	'Branch out of range'	
1953	EEED	04		FCB	EOT	
1954	EEEE		ASSEM	EQU	*	
1955	EEEE	CE 00 00		LDX	#RAMBS	
1956	EEF1	DF 7F		STX	OLDPC	
1957	EEF3	BD E3 8D		JSR	WSKIP	
1958	EEF6	81 0D		CMPA	#SD	
1959	EEF8	27 15		BEQ	ASSLOOP	jump if no argument
1960	EEFA	BD E2 B2		JSR	BUFFARG	
1961	EEFD	BD E3 8D		JSR	WSKIP	
1962	EF00	81 0D		CMPA	#SD	
1963	EF02	27 07		BEQ	ASSEM1	jump if argument ok
1964	EF04	CE EE 7A		LDX	#MSGAS4	"bad argument"
1965	EF07	BD E2 08		JSR	OUTSTRG	
1966	EF0A	39		RTS		
1967	EF0B	DE 68	ASSEM1	LDX	SHFTREG	
1968	EF0D	DF 7F		STX	OLDPC	
1969						
1970	EF0F	DE 7F	ASSLOOP	LDX	OLDPC	
1971	EF11	DF 83		STX	PC	
1972	EF13	BD E1 F7		JSR	OUTCRLF	
1973	EF16	BD E2 4E		JSR	CLRBUFF	clear display buffer
1974	EF19	CE 00 83		LDX	#PC	
1975	EF1C	BD E4 1B		JSR	OUT2BSP	output the address
1976	EF1F	BD E4 21		JSR	OUTSPAC	formatting
1977	EF22	BD F7 57		JSR	DISASSM	disassemble opcode
1978	EF25	BD E1 F7		JSR	OUTCRLF	
1979	EF28	7D 00 01		TST	IODEV	check device type
1980	EF2B	27 03		BEQ	ASSLOOP1	if SCI take branch
1981	EF2D	BD F9 EB		JSR	VIDEO	wait for input

1982	EF30	BD E4 21	ASSLOOP1	JSR	OUTSPAC	
1983	EF33	BD E4 21		JSR	OUTSPAC	
1984	EF36	BD E4 21		JSR	OUTSPAC	
1985	EF39	BD E4 21		JSR	OUTSPAC	
1986	EF3C	BD E2 4E		JSR	CLRBUFF	
1987	EF3F	BD EF D8		JSR	READLN	read input for assembly
1988	EF42	97 8F		STAA	ASSCOMM	
1989	EF44	81 5E		CMPA	#'^	
1990	EF46	27 09		BEQ	ASSLP0	jump if up arrow
1991	EF48	81 2F		CMPA	#/'	
1992	EF4A	27 05		BEQ	ASSLP0	jump if slash
1993	EF4C	81 00		CMPA	#\$00	
1994	EF4E	26 04		BNE	ASSLP1	jump if none of above
1995	EF50	39		RTS		return if bad input
1996	EF51	BD E1 F7	ASSLP0	JSR	OUTCRLF	
1997	EF54	7D 00 01	ASSLP1	TST	IODEV	
1998	EF57	26 0F		BNE	ASSLP01	format for SPI
1999	EF59	BD E4 21		JSR	OUTSPAC	
2000	EF5C	BD E4 21		JSR	OUTSPAC	
2001	EF5F	BD E4 21		JSR	OUTSPAC	
2002	EF62	BD E4 21		JSR	OUTSPAC	
2003	EF65	BD E4 21		JSR	OUTSPAC	
2004	EF68	BD F0 33	ASSLP01	JSR	PARSE	
2005	EF6B	C1 05		CMPB	#\$5	
2006	EF6D	2F 08		BLE	ASSLP2	jump if mnemonic <= 5 chars
2007	EF6F	CE EE 87		LDX	#\$MSGA5	"mnemonic not found"
2008	EF72	BD E2 08		JSR	OUTSTRG	
2009	EF75	20 2F		BRA	ASSLP5	
2010	EF77		ASSLP2	EQU	*	
2011	EF77	C1 00		CMPB	#\$0	
2012	EF79	27 2E		BEQ	ASSLP10	jump if no input
2013	EF7B	BD F0 74		JSR	MSRCH	
2014	EF7E	96 88		LDAA	CLASS	
2015	EF80	81 FF		CMPA	#\$FF	
2016	EF82	26 08		BNE	ASSLP3	
2017	EF84	CE EE 87		LDX	#\$MSGA5	"mnemonic not found"
2018	EF87	BD E2 08		JSR	OUTSTRG	
2019	EF8A	20 1A		BRA	ASSLP5	
2020	EF8C	BD F0 AC	ASSLP3	JSR	DOOP	
2021	EF8F	81 00		CMPA	#\$00	
2022	EF91	26 07		BNE	ASSLP4	jump if doop error
2023	EF93	CE 00 00		LDX	#\$00	
2024	EF96	DF 89		STX	DISPC	indicate good assembly
2025	EF98	20 0F		BRA	ASSLP10	
2026	EF9A	4A	ASSLP4	DECA		a = error message index
2027	EF9B	16		TAB		
2028	EF9C	CE EE 2A		LDX	#\$MSGDIR	
2029	EF9F	3A		ABX		
2030	EFA0	3A		ABX		
2031	EFA1	EE 00		LDX	0,X	
2032	EFA3	BD E2 08		JSR	OUTSTRG	output error message

2033	EFA6	7F 00 8F	ASSLP5	CLR	ASSCOMM	error command
2034	EFA9		ASSLP10	EQU	*	
2035	EFA9	96 8F		LDAA	ASSCOMM	
2036	EFAB	81 5E		CMPA	#'^	
2037	EFAD	26 07		BNE	ASSLP11	jump if not up arrow
2038	EFAD	DE 7F		LDX	OLDPC	
2039	EFB1	09		DEX		
2040	EFB2	DF 7F		STX	OLDPC	back up
2041	EFB4	20 1F		BRA	ASSLP15	
2042	EFB6	81 0A	ASSLP11	CMPA	#\$0A	
2043	EFB8	27 04		BEQ	ASSLP12	jump if linefeed
2044	EFBA	81 0D		CMPA	#\$0D	
2045	EFBC	26 17		BNE	ASSLP15	jump if not cr
2046	EFBE	DE 89	ASSLP12	LDX	DISPC	
2047	EFC0	26 06		BNE	ASSLP13	jump if dispc != 0
2048	EFC2	DE 83		LDX	PC	
2049	EFC4	DF 7F		STX	OLDPC	
2050	EFC6	20 0D		BRA	ASSLP15	
2051	EFC8	81 0A	ASSLP13	CMPA	#\$0A	
2052	EFCB	26 05		BNE	ASSLP14	jump if not linefeed
2053	EFCC	DE 7F		LDX	OLDPC	
2054	EFCE	08		INX		
2055	EFCF	DF 89		STX	DISPC	
2056	EFD1	DE 89	ASSLP14	LDX	DISPC	
2057	EFD3	DF 7F		STX	OLDPC	
2058	EFD5	7E EF 0F	ASSLP15	JMP	ASSLOOP	
2059						
2060						
2061						
2062						
2063						
2064						
2065						
2066						
2067						
2068						
2069						
2070	EFD8	BD E2 4E	READLN	JSR	CLRBUFF	
2071	EFDB	86 2E		LDAA	#.'	
2072	EFDD	BD E1 76		JSR	OUTPUT	output asm prompt character
2073	EFE0	BD E1 37	RLN1	JSR	INCHAR	
2074	EFE3	BD E2 74		JSR	UPCASE	
2075	EFE6	CE 00 05		LDX	#CHARBUFF	
2076	EFE9	D6 02		LDAB	CHARPTR	
2077	EFEB	3A		ABX		
2078	EFEC	81 7F		CMPA	#DEL	Delete
2079	EFEE	27 41		BEQ	RLNQUIT	
2080	EFF0	81 18		CMPA	#CTLX	Control X
2081	EFF2	27 3D		BEQ	RLNQUIT	
2082	EFF4	81 02		CMPA	#CTLA	Control A
2083	EFF6	27 39		BEQ	RLNQUIT	

* readln() --- Read input from terminal into buffer

* until a command character is read (cr,lf,/,^).

* If more chars are typed than the buffer will hold,

* the extra characters are overwritten on the end.

* On exit: b=number of chars read, a=0 if quit,

* else a=next command.

2084	EFF8	81 03		CMPA #S03	Control C
2085	EFFA	27 35		BEQ RLNQUIT	
2086	EFFC	81 04		CMPA #S04	Control D
2087	EFFE	27 31		BEQ RLNQUIT	
2088	F000	81 08		CMPA #S08	backspace
2089	F002	26 11		BNE RLN2	
2090	F004	7D 00 01		TST IODEV	
2091	F007	27 05		BEQ RDLN10	
2092	F009	BD E1 64		JSR CRTBKSPC	backspace on CHARBUFF
2093	F00C	20 D2		BRA RLN1	get another character
2094	F00E	7A 00 02	RDLN10	DEC CHARPTR	
2095	F011	2C CD		BGE RLN1	
2096	F013	20 C3		BRA READLN	start over
2097	F015	7D 00 01	RLN2	TST IODEV	
2098	F018	26 05		BNE RLN21	
2099	F01A	A7 00		STAA 0,X	put char in buffer
2100	F01C	5C		INCB	point to next pos in buffer
2101	F01D	D7 02		STAB CHARPTR	update pointer
2102	F01F	C1 1E	RLN21	CMPB #BUFFLEN	max buffer length
2103	F021	2D 08		BLT RLN3	jump if buffer not full
2104	F023	CE E4 97		LDX #MSG3	'too long'
2105	F026	BD E2 08		JSR OUTSTRG	
2106	F029	20 AD		BRA READLN	
2107	F02B	BD F0 65	RLN3	JSR ASSCHEK	check for subcommand
2108	F02E	26 B0		BNE RLN1	
2109	F030	39		RTS	
2110	F031	4F	RLNQUIT	CLRA	quit
2111	F032	39		RTS	return
2112					
2113				*****	
2114				* parse() -parse out the mnemonic from CHARBUFF	
2115				* to COMBUFF. on exit: b=number of chars parsed.	
2116				*****	
2117					
2118	F033	86 20	PARSE	LDAA #S20	
2119	F035	97 63		STAA COMBUFF+3	
2120	F037	CE 00 05		LDX #CHARBUFF	initialize buffer ptr
2121	F03A	7D 00 01		TST IODEV	
2122	F03D	27 01		BEQ PARS1	
2123	F03F	08		INX	adjust pointer if SPI
2124	F040	DF 81	PARS1	STX PTR0	
2125	F042	BD E3 8D		JSR WSKIP	find first character
2126	F045	5F		CLRB	
2127	F046	BD E3 78	PARSLP	JSR RDBUFF	read character
2128	F049	BD E3 7F		JSR INCBUFF	
2129	F04C	BD E3 9B		JSR WCHEK	
2130	F04F	27 13		BEQ PARSRT	jump if whitespace
2131	F051	BD F0 65		JSR ASSCHEK	
2132	F054	27 0E		BEQ PARSRT	jump if end of line
2133	F056	BD E2 74		JSR UPCASE	convert to upper case

```

2134 F059 CE 00 60          LDX  #COMBUFF
2135 F05C 3A              ABX
2136 F05D A7 00          STAA 0,X          store in combuff
2137 F05F 5C              INCB
2138 F060 C1 05          CMPB  #5
2139 F062 2F E2          BLE  PARSLP      loop 6 times
2140 F064 39          PARSRT  RTS
2141
2142          *****
2143          * asschek() -perform compares for
2144          * cr, lf, ^, /
2145          *****
2146
2147 F065 81 0A          ASSCHEK  CMPA  #0A          linefeed
2148 F067 27 0A          BEQ  ASSCHK1
2149 F069 81 0D          CMPA  #0D          carriage ret
2150 F06B 27 06          BEQ  ASSCHK1
2151 F06D 81 5E          CMPA  #'^'          up arrow
2152 F06F 27 02          BEQ  ASSCHK1
2153 F071 81 2F          CMPA  #'/'          slash
2154 F073 39          ASSCHK1  RTS
2155
2156          *****
2157          * msrch() --- Search MNETABL for mnemonic in COMBUFF.
2158          * stores base opcode at baseop and class at class.
2159          * Class = FF if not found.
2160          *****
2161
2162 F074 CE F3 D8          MSRCH  LDX  #MNETABL      pointer to mnemonic table
2163 F077 18 CE 00 60          LDY  #COMBUFF      pointer to string
2164 F07B 20 03              BRA  MSRCH1
2165 F07D              MSNEXT  EQU  *
2166 F07D C6 06          LDAB  #6
2167 F07F 3A              ABX          point to next table entry
2168 F080 A6 00          MSRCH1  LDAA 0,X          read table
2169 F082 81 04          CMPA  #EOT
2170 F084 26 05          BNE  MSRCH2      jump if not end of table
2171 F086 86 FF          LDAA  #FFF
2172 F088 97 88          STAA  CLASS      FF = not in table
2173 F08A 39              RTS
2174 F08B 18 A1 00          MSRCH2  CMPA 0,Y          op[0] = tabl[0] ?
2175 F08E 26 ED          BNE  MSNEXT
2176 F090 A6 01          LDAA 1,X
2177 F092 18 A1 01          CMPA 1,Y          op[1] = tabl[1] ?
2178 F095 26 E6          BNE  MSNEXT
2179 F097 A6 02          LDAA 2,X
2180 F099 18 A1 02          CMPA 2,Y          op[2] = tabl[2] ?
2181 F09C 26 DF          BNE  MSNEXT
2182 F09E A6 03          LDAA 3,X
2183 F0A0 18 A1 03          CMPA 3,Y          op[2] = tabl[2] ?
2184 F0A3 26 D8          BNE  MSNEXT

```

2185	F0A5	EC 04		LDD	4,X	opcode, class
2186	F0A7	97 87		STAA	BASEOP	
2187	F0A9	D7 88		STAB	CLASS	
2188	F0AB	39		RTS		
2189						
2190						
2191				** doop(baseop,class) — process mnemonic.		
2192				** on exit: a=error code corresponding to error		
2193				** messages.		
2194						
2195						
2196	F0AC		DOOP	EQU	*	
2197	F0AC	86 04		LDAA	#OTHER	
2198	F0AE	97 92		STAA	AMODE	mode
2199	F0B0	7F 00 93		CLR	YFLAG	
2200	F0B3	DE 81		LDX	PTR0	
2201	F0B5	8C 00 23	DOPLP1	CPX	#ENDBUFF	(end of buffer)
2202	F0B8	27 1B		BEQ	DOOP1	jump if end of buffer
2203	F0BA	EC 00		LDD	0,X	read 2 chars from buffer
2204	F0BC	08		INX		move pointer
2205	F0BD	81 2C		CMPA	#','	
2206	F0BF	26 F4		BNE	DOPLP1	
2207	F0C1	C1 59		CMPB	#'Y'	look for ",y"
2208	F0C3	26 06		BNE	DOPLP2	
2209	F0C5	86 02		LDAA	#INDY	
2210	F0C7	97 92		STAA	AMODE	
2211	F0C9	20 0A		BRA	DOOP1	
2212	F0CB	C1 58	DOPLP2	CMPB	#'X'	look for ",x"
2213	F0CD	26 06		BNE	DOOP1	jump if not x
2214	F0CF	86 01		LDAA	#INDX	
2215	F0D1	97 92		STAA	AMODE	
2216	F0D3	20 00		BRA	DOOP1	
2217	F0D5	BD E3 8D	DOOP1	JSR	WSKIP	
2218	F0D8	81 23		CMPA	#'#'	look for immediate mode
2219	F0DA	26 07		BNE	DOOP2	
2220	F0DC	BD E3 7F		JSR	INCBUFF	point at argument
2221	F0DF	86 00		LDAA	#IMMED	
2222	F0E1	97 92		STAA	AMODE	
2223	F0E3	D6 88	DOOP2	LDAB	CLASS	
2224	F0E5	C1 02		CMPB	#P2INH	
2225	F0E7	26 03		BNE	DOSW1	
2226	F0E9	7E F1 4A		JMP	DOP2I	
2227	F0EC	C1 01	DOSW1	CMPB	#INH	
2228	F0EE	26 03		BNE	DOSW2	
2229	F0F0	7E F1 4F		JMP	DOINH	
2230	F0F3	C1 05	DOSW2	CMPB	#REL	
2231	F0F5	26 03		BNE	DOSW3	
2232	F0F7	7E F1 56		JMP	DOREL	
2233	F0FA	C1 08	DOSW3	CMPB	#LIMM	
2234	F0FC	26 03		BNE	DOSW4	
2235	F0FE	7E F1 85		JMP	DOLIM	

2236	F101	C1 07	DOSW4	CMPB #NIMM	
2237	F103	26 03		BNE DOSW5	
2238	F105	7E F1 8F		JMP DONOI	
2239	F108	C1 03	DOSW5	CMPB #GEN	
2240	F10A	26 03		BNE DOSW6	
2241	F10C	7E F1 98		JMP DOGENE	
2242	F10F	C1 04	DOSW6	CMPB #GRP2	
2243	F111	26 03		BNE DOSW7	
2244	F113	7E F1 A6		JMP DOGRP	
2245	F116	C1 15	DOSW7	CMPB #CPD	
2246	F118	26 03		BNE DOSW8	
2247	F11A	7E F1 D8		JMP DOCPD	
2248	F11D	C1 10	DOSW8	CMPB #XNIMM	
2249	F11F	26 03		BNE DOSW9	
2250	F121	7E F1 F9		JMP DOXNOI	
2251	F124	C1 09	DOSW9	CMPB #XLIMM	
2252	F126	26 03		BNE DOSW10	
2253	F128	7E F2 02		JMP DOXLI	
2254	F12B	C1 12	DOSW10	CMPB #YNIMM	
2255	F12D	26 03		BNE DOSW11	
2256	F12F	7E F2 1A		JMP DOYNOI	
2257	F132	C1 11	DOSW11	CMPB #YLIMM	
2258	F134	26 03		BNE DOSW12	
2259	F136	7E F2 23		JMP DOYLI	
2260	F139	C1 13	DOSW12	CMPB #BTB	
2261	F13B	26 03		BNE DOSW13	
2262	F13D	7E F2 42		JMP DOBTB	
2263	F140	C1 14	DOSW13	CMPB #SETCLR	
2264	F142	26 03		BNE DODEF	
2265	F144	7E F2 42		JMP DOSET	
2266	F147	86 02	DODEF	LDAA #\$2	
2267	F149	39		RTS	
2268	F14A	86 18	DOP2I	LDAA #PAGE2	
2269	F14C	BD F3 CB		JSR EMIT	
2270	F14F	96 87	DOINH	LDAA BASEOP	
2271	F151	BD F3 CB		JSR EMIT	
2272	F154	4F		CLRA	
2273	F155	39		RTS	
2274	F156	BD F3 AD	DOREL	JSR ASSARG	
2275	F159	81 04		CMPA #\$04	
2276	F15B	26 01		BNE DOREL1	jump if arg ok
2277	F15D	39		RTS	
2278	F15E	DC 68	DOREL1	LDD SHFTREG	get branch address
2279	F160	DE 83		LDX PC	get program counter
2280	F162	08		INX	
2281	F163	08		INX	point to end of opcode
2282	F164	DF 8B		STX BRADDR	
2283	F166	93 8B		SUBD BRADDR	calculate offset
2284	F168	DD 8B		STD BRADDR	save result
2285	F16A	1A 83 00 7F		CMPD #\$7F	in range ?
2286	F16E	23 09		BLS DOREL2	jump if in range

2287	F170	1A 83 FF 80		CMPD #FFF80	
2288	F174	24 03		BHS DOREL2	jump if in range
2289	F176	86 09		LDAA #\$09	'Out of range'
2290	F178	39		RTS	
2291	F179	97 87	DOREL2	LDAA BASEOP	
2292	F17B	BD F3 CB		JSR EMIT	emit opcode
2293	F17E	96 8C		LDAA BRADDR+1	
2294	F180	BD F3 CB		JSR EMIT	emit offset
2295	F183	4F		CLRA	normal return
2296	F184	39		RTS	
2297	F185	96 92	DOLIM	LDAA AMODE	
2298	F187	81 00		CMPA #IMMED	
2299	F189	26 04		BNE DONOI	
2300	F18B	86 03		LDAA #LIMMED	
2301	F18D	97 92		STAA AMODE	
2302	F18F	96 92	DONOI	LDAA AMODE	
2303	F191	81 00		CMPA #IMMED	
2304	F193	26 03		BNE DOGENE	jump if not immediate
2305	F195	86 01		LDAA #\$1	"immediate mode illegal"
2306	F197	39		RTS	
2307	F198	86 00	DOGENE	LDAA #PAGE1	
2308	F19A	97 94		STAA PNORM	
2309	F19C	97 85		STAA PX	
2310	F19E	86 18		LDAA #PAGE2	
2311	F1A0	97 86		STAA PY	
2312	F1A2	BD F2 DE		JSR DOGEN	
2313	F1A5	39		RTS	
2314	F1A6	96 92	DOGRP	LDAA AMODE	
2315	F1A8	81 02		CMPA #INDY	
2316	F1AA	26 09		BNE DOGRP1	
2317	F1AC	86 18		LDAA #PAGE2	
2318	F1AE	BD F3 CB		JSR EMIT	
2319	F1B1	86 01		LDAA #INDX	
2320	F1B3	97 92		STAA AMODE	
2321	F1B5		DOGRP1	EQU *	
2322	F1B5	96 92		LDAA AMODE	
2323	F1B7	81 01		CMPA #INDX	
2324	F1B9	26 04		BNE DOGRP2	
2325	F1BB	BD F3 7E		JSR DOINDEX	
2326	F1BE	39		RTS	
2327	F1BF		DOGRP2	EQU *	
2328	F1BF	96 87		LDAA BASEOP	
2329	F1C1	8B 10		ADDA #\$10	
2330	F1C3	BD F3 CB		JSR EMIT	
2331	F1C6	BD F3 AD		JSR ASSARG	
2332	F1C9	81 04		CMPA #\$04	
2333	F1CB	27 0A		BEQ DOGRPRT	jump if bad arg
2334	F1CD	DC 68		LDD SHFTREG	extended address
2335	F1CF	BD F3 CB		JSR EMIT	
2336	F1D2	17		TBA	
2337	F1D3	BD F3 CB		JSR EMIT	

2338	F1D6	4F		CLRA	
2339	F1D7	39	DOGRPRT	RTS	
2340	F1D8	96 92	DOCPD	LDAA AMODE	
2341	F1DA	81 00		CMPA #IMMED	
2342	F1DC	26 04		BNE DOCPD1	
2343	F1DE	86 03		LDAA #LIMMED	
2344	F1E0	97 92		STAA AMODE	
2345	F1E2	96 92	DOCPD1	LDAA AMODE	
2346	F1E4	81 02		CMPA #INDY	
2347	F1E6	26 03		BNE DOCPD2	
2348	F1E8	7C 00 93		INC YFLAG	
2349	F1EB	86 1A	DOCPD2	LDAA #PAGE3	
2350	F1ED	97 94		STAA PNORM	
2351	F1EF	97 85		STAA PX	
2352	F1F1	86 CD		LDAA #PAGE4	
2353	F1F3	97 86		STAA PY	
2354	F1F5	BD F2 DE		JSR DOGEN	
2355	F1F8	39		RTS	
2356	F1F9	96 92	DOXNOI	LDAA AMODE	
2357	F1FB	81 00		CMPA #IMMED	
2358	F1FD	26 03		BNE DOXLI	
2359	F1FF	86 01		LDAA #\$1	"immediate mode illegal"
2360	F201	39		RTS	
2361	F202	96 92	DOXLI	LDAA AMODE	
2362	F204	81 00		CMPA #IMMED	
2363	F206	26 04		BNE DOXLI1	
2364	F208	86 03		LDAA #LIMMED	
2365	F20A	97 92		STAA AMODE	
2366	F20C	86 00	DOXLI1	LDAA #PAGE1	
2367	F20E	97 94		STAA PNORM	
2368	F210	97 85		STAA PX	
2369	F212	86 CD		LDAA #PAGE4	
2370	F214	97 86		STAA PY	
2371	F216	BD F2 DE		JSR DOGEN	
2372	F219	39		RTS	
2373	F21A	96 92	DOYNOI	LDAA AMODE	
2374	F21C	81 00		CMPA #IMMED	
2375	F21E	26 03		BNE DOYLI	
2376	F220	86 01		LDAA #\$1	"immediate mode illegal"
2377	F222	39		RTS	
2378	F223	96 92	DOYLI	LDAA AMODE	
2379	F225	81 02		CMPA #INDY	
2380	F227	26 03		BNE DOYLI1	
2381	F229	7C 00 93		INC YFLAG	
2382	F22C	81 00	DOYLI1	CMPA #IMMED	
2383	F22E	26 04		BNE DOYLI2	
2384	F230	86 03		LDAA #LIMMED	
2385	F232	97 92		STAA AMODE	
2386	F234	86 18	DOYLI2	LDAA #PAGE2	
2387	F236	97 94		STAA PNORM	
2388	F238	97 86		STAA PY	

2389	F23A	86 1A		LDAA #PAGE3	
2390	F23C	97 85		STAA PX	
2391	F23E	BD F2 DE		JSR DOGEN	
2392	F241	39		RTS	
2393	F242		DOBTB	EQU *	
2394	F242	BD F2 B8	DOSET	JSR BITOP	
2395	F245	81 00		CMPA #\$00	
2396	F247	26 03		BNE DOSET1	
2397	F249	86 03		LDAA #\$3	"illegal bit op"
2398	F24B	39		RTS	
2399	F24C	96 92	DOSET1	LDAA AMODE	
2400	F24E	81 02		CMPA #INDY	
2401	F250	26 09		BNE DOSET2	
2402	F252	86 18		LDAA #PAGE2	
2403	F254	BD F3 CB		JSR EMIT	
2404	F257	86 01		LDAA #INDX	
2405	F259	97 92		STAA AMODE	
2406	F25B	96 87	DOSET2	LDAA BASEOP	
2407	F25D	BD F3 CB		JSR EMIT	
2408	F260	BD F3 AD		JSR ASSARG	
2409	F263	81 04		CMPA #\$04	
2410	F265	26 01		BNE DOSET22	jump if arg ok
2411	F267	39		RTS	
2412	F268	96 69	DOSET22	LDAA SHFTREG+1	index offset
2413	F26A	BD F3 CB		JSR EMIT	
2414	F26D	96 92		LDAA AMODE	
2415	F26F	81 01		CMPA #INDX	
2416	F271	26 06		BNE DOSET3	
2417	F273	BD E3 7F		JSR INCBUFF	
2418	F276	BD E3 7F		JSR INCBUFF	
2419	F279	BD F3 AD	DOSET3	JSR ASSARG	
2420	F27C	81 04		CMPA #\$04	
2421	F27E	26 01		BNE DOSET33	jump if arg ok
2422	F280	39		RTS	
2423	F281	96 69	DOSET33	LDAA SHFTREG+1	mask
2424	F283	BD F3 CB		JSR EMIT	
2425	F286	96 88		LDAA CLASS	
2426	F288	81 14		CMPA #SETCLR	
2427	F28A	26 02		BNE DOSET4	
2428	F28C	4F		CLRA	
2429	F28D	39		RTS	
2430	F28E	BD F3 AD	DOSET4	JSR ASSARG	
2431	F291	81 04		CMPA #\$04	
2432	F293	26 01		BNE DOSET5	jump if arg ok
2433	F295	39		RTS	
2434	F296	DE 83	DOSET5	LDX PC	program counter
2435	F298	08		INX	point to next inst
2436	F299	DF 8B		STX BRADDR	save pc value
2437	F29B	DC 68		LDD SHFTREG	get branch address
2438	F29D	93 8B		SUBD BRADDR	
			calculate offset		

```

2439 F29F 1A 83 00 7F      CMPD #$7F
2440 F2A3 23 0D            BLS  DOSET6      jump if in range
2441 F2A5 1A 83 FF 80      CMPD #$FF80
2442 F2A9 24 07            BHS  DOSET6      jump if in range
2443 F2AB 4F              CLRA
2444 F2AC BD F3 CB          JSR  EMIT
2445 F2AF 86 09            LDAA #$09        'out of range'
2446 F2B1 39              RTS
2447 F2B2 17              TBA              offset
2448 F2B3 BD F3 CB          JSR  EMIT
2449 F2B6 4F              CLRA
2450 F2B7 39              RTS
2451
2452
2453      *****
2454      ** bitop(baseop,amode,class) --- adjust opcode on bit
2455      ** manipulation instructions. Returns opcode in a
2456      ** or a = 0 if error
2457      *****
2458 F2B8 96 92      BITOP      LDAA AMODE
2459 F2BA D6 88      LDAB CLASS
2460 F2BC 81 01      CMPA #INDX
2461 F2BE 26 01      BNE  BITOP1
2462 F2C0 39          RTS
2463 F2C1 81 02      BITOP1     CMPA #INDY
2464 F2C3 26 01      BNE  BITOP2      jump not indexed
2465 F2C5 39          RTS
2466 F2C6 C1 14      BITOP2     CMPB #SETCLR
2467 F2C8 26 07      BNE  BITOP3      jump not bset,bclr
2468 F2CA 96 87      LDAA BASEOP      get opcode
2469 F2CC 80 08      SUBA #8
2470 F2CE 97 87      STAA BASEOP
2471 F2D0 39          RTS
2472 F2D1 C1 13      BITOP3     CMPB #BTB
2473 F2D3 26 07      BNE  BITOP4      jump not bit branch
2474 F2D5 96 87      LDAA BASEOP      get opcode
2475 F2D7 80 0C      SUBA #12
2476 F2D9 97 87      STAA BASEOP
2477 F2DB 39          RTS
2478 F2DC 4F          BITOP4     CLRA      0 = fatal bitop
2479 F2DD 39          RTS
2480
2481      *****
2482      ** dogen(baseop,mode,pnorm,px,py) - process
2483      ** general addressing modes. Returns a = error #.
2484      *****
2485
2486 F2DE 96 92      DOGEN      LDAA AMODE
2487 F2E0 81 03      CMPA #LIMMED
2488 F2E2 27 13      BEQ  DOGLIM
2489 F2E4 81 00      CMPA #IMMED

```


2490	F2E6	27 2C		BEQ	DOGIMM	
2491	F2E8	81 02		CMPA	#INDY	
2492	F2EA	27 41		BEQ	DOGINDY	
2493	F2EC	81 01		CMPA	#INDX	
2494	F2EE	27 4C		BEQ	DOGINDX	
2495	F2F0	81 04		CMPA	#OTHER	
2496	F2F2	27 57		BEQ	DOGOTH	
2497	F2F4	86 06	DOGDEF	LDAA	#\$06	unknown addre...
2498	F2F6	39		RTS		
2499	F2F7	96 94	DOGLIM	LDAA	PNORM	
2500	F2F9	BD F3 C3		JSR	EPAGE	
2501	F2FC	96 87	DOGLIM1	LDAA	BASEOP	
2502	F2FE	BD F3 CB		JSR	EMIT	
2503	F301	BD F3 AD		JSR	ASSARG	get next argument
2504	F304	81 04		CMPA	#\$04	
2505	F306	26 01		BNE	DOGLIM2	jump if arg ok
2506	F308	39		RTS		
2507	F309	DC 68	DOGLIM2	LDD	SHFTREG	
2508	F30B	BD F3 CB		JSR	EMIT	
2509	F30E	17		TBA		
2510	F30F	BD F3 CB		JSR	EMIT	
2511	F312	4F		CLRA		
2512	F313	39		RTS		
2513	F314	96 94	DOGIMM	LDAA	PNORM	
2514	F316	BD F3 C3		JSR	EPAGE	
2515	F319	96 87		LDAA	BASEOP	
2516	F31B	BD F3 CB		JSR	EMIT	
2517	F31E	BD F3 AD		JSR	ASSARG	
2518	F321	81 04		CMPA	#\$04	
2519	F323	26 01		BNE	DOGIMM1	jump if arg ok
2520	F325	39		RTS		
2521	F326	96 69	DOGIMM1	LDAA	SHFTREG+1	
2522	F328	BD F3 CB		JSR	EMIT	
2523	F32B	4F		CLRA		
2524	F32C	39		RTS		
2525	F32D	96 86	DOGINDY	LDAA	PY	
2526	F32F	BD F3 C3		JSR	EPAGE	
2527	F332	96 87		LDAA	BASEOP	
2528	F334	8B 20		ADDA	#\$20	
2529	F336	97 87		STAA	BASEOP	
2530	F338	BD F3 7E		JSR	DOINDEX	
2531	F33B	39		RTS		
2532	F33C	96 85	DOGINDX	LDAA	PX	
2533	F33E	BD F3 C3		JSR	EPAGE	
2534	F341	96 87		LDAA	BASEOP	
2535	F343	8B 20		ADDA	#\$20	
2536	F345	97 87		STAA	BASEOP	
2537	F347	BD F3 7E		JSR	DOINDEX	
2538	F34A	39		RTS		
2539	F34B	BD F3 AD	DOGOTH	JSR	ASSARG	
2540	F34E	81 04		CMPA	#\$04	

2541	F350	26 01		BNE	DOGOTH0	jump if arg ok
2542	F352	39		RTS		
2543	F353	96 94	DOGOTH0	LDAA	PNORM	
2544	F355	BD F3 C3		JSR	EPAGE	
2545	F358	96 7E		LDAA	COUNT	
2546	F35A	81 02		CMPA	#\$2	
2547	F35C	2E 0E		BGT	DOGOTH1	
2548	F35E	96 87		LDAA	BASEOP	
2549	F360	8B 10		ADDA	#\$10	direct mode opcode
2550	F362	BD F3 CB		JSR	EMIT	
2551	F365	96 69		LDAA	SHFTREG+1	
2552	F367	BD F3 CB		JSR	EMIT	
2553	F36A	4F		CLRA		
2554	F36B	39		RTS		
2555	F36C	96 87	DOGOTH1	LDAA	BASEOP	
2556	F36E	8B 30		ADDA	#\$30	extended mode opcode
2557	F370	BD F3 CB		JSR	EMIT	
2558	F373	DC 68		LDD	SHFTREG	
2559	F375	BD F3 CB		JSR	EMIT	
2560	F378	17		TBA		
2561	F379	BD F3 CB		JSR	EMIT	
2562	F37C	4F		CLRA		
2563	F37D	39		RTS		
2564						
2565						
2566				** doindex(op) --- handle all wierd stuff for		
2567				** indexed addressing. Returns a = error number.		
2568						
2569						
2570	F37E	96 87	DOINDEX	LDAA	BASEOP	
2571	F380	BD F3 CB		JSR	EMIT	
2572	F383	BD F3 AD		JSR	ASSARG	
2573	F386	81 04		CMPA	#\$04	
2574	F388	26 01		BNE	DOINDX0	jump if arg ok
2575	F38A	39		RTS		
2576	F38B	81 2C	DOINDX0	CMPA	#','	
2577	F38D	27 03		BEQ	DOINDX1	
2578	F38F	86 08		LDAA	#\$08	"syntax error"
2579	F391	39		RTS		
2580	F392	BD E3 7F	DOINDX1	JSR	INCBUFF	
2581	F395	BD E3 78		JSR	RDBUFF	
2582	F398	81 59		CMPA	#'Y'	
2583	F39A	27 0A		BEQ	DOINDX2	
2584	F39C	81 58		CMPA	#'X'	
2585	F39E	27 06		BEQ	DOINDX2	
2586	F3A0	FE EE B2		LDD	MSG A7	"index addr assumed"
2587	F3A7	BD E2 08		JSR	OUTSTRG	
2588	F3A6	96 69	DOINDX2	LDAA	SHFTREG+1	
2589	F3A8	BD F3 CB		JSR	EMIT	
2590	F3AB	4F		CLRA		
2591	F3AC	39		RTS		

```

2592
2593 *****
2594 ** assarg(); - get argument. Returns a = 4 if bad
2595 ** argument, else a = first non hex char.
2596 *****
2597
2598 F3AD BD E2 B2 ASSARG JSR BUFFARG
2599 F3B0 BD F0 65 JSR ASSCHEK check for command
2600 F3B3 27 05 BEQ ASSARG1 jump if ok
2601 F3B5 BD E3 9B JSR WCHEK check for whitespace
2602 F3B8 26 06 BNE ASSARG2 jump if not ok
2603 F3BA 7D 00 7E ASSARG1 TST COUNT
2604 F3BD 27 01 BEQ ASSARG2 jump if no argument
2605 F3BF 39 RTS
2606 F3C0 86 04 ASSARG2 LDAA #$04 bad argument
2607 F3C2 39 RTS
2608
2609 *****
2610 ** epage(a) --- emit page prebyte
2611 *****
2612
2613 F3C3 81 00 EPAGE CMPA #PAGE1
2614 F3C5 27 03 BEQ EPAGRT jump if page 1
2615 F3C7 BD F3 CB JSR EMIT
2616 F3CA 39 EPAGRT RTS
2617
2618 *****
2619 * emit(a) --- emit contents of a
2620 *****
2621
2622 F3CB DE 83 EMIT LDX PC
2623 F3CD 97 69 STAA SHFTREG+1 so CHGBYT0 works
2624 F3CF BD E3 04 JSR CHGBYT0 put data in memory
2625 F3D2 BD E4 1E JSR OUT1BSP
2626 F3D5 DF 83 STX PC
2627 F3D7 39 RTS
2628
2629
2630 0000 NULL EQU $0 *Mnemonic table for hc11 line
2631 0001 INH EQU $1 assembler
2632 0002 P2INH EQU $2 nothing
2633 0003 GEN EQU $3 inherent
2634 0004 GRP2 EQU $4 page 2 inherent
2635 0005 REL EQU $5 general addressing
2636 0006 IMM EQU $6 group 2
2637 0007 NIMM EQU $7 relative
2638 0008 LIMM EQU $8 immediate
2639 0009 XLIMM EQU $9 general except for immediate
2640 0010 XNIMM EQU $10 2 byte immediate
2641 0011 YLIMM EQU $11 longimm for x
longimm for y

```

2642	0012	YNIMM	EQU	\$12	no immediate for y
2643	0013	BTB	EQU	\$13	bit test and branch
2644	0014	SETCLR	EQU	\$14	bit set or clear
2645	0015	CPD	EQU	\$15	compare d
2646	0016	BTBD	EQU	\$16	bit test and branch direct
2647	0017	SETCLR D	EQU	\$17	bit set or clear direct

2648

2649

2650

2651

2652

2653

2654

2655

* mnetabl - includes all '11 mnemonics, base opcodes,

* and type of instruction. The assembler search routine

* depends on 4 characters for each mnemonic so that 3 char

* mnemonics are extended with a space and 5 char mnemonics

* are truncated.

2656 F3D8

MNETABL

EQU

*

2657 F3D8 41 42 41 20

FCC

'ABA'

Mnemonic

2658 F3DC 1B

FCB

\$1B

Base opcode

2659 F3DD 01

FCB

INH

Class

2660 F3DE 41 42 58 20

FCC

'ABX'

2661 F3E2 3A

FCB

\$3A

2662 F3E3 01

FCB

INH

2663 F3E4 41 42 59 20

FCC

'ABY'

2664 F3E8 3A

FCB

\$3A

2665 F3E9 02

FCB

P2INH

2666 F3EA 41 44 43 41

FCC

'ADCA'

2667 F3EE 89

FCB

\$89

2668 F3EF 03

FCB

GEN

2669 F3F0 41 44 43 42

FCC

'ADCB'

2670 F3F4 C9

FCB

\$C9

2671 F3F5 03

FCB

GEN

2672 F3F6 41 44 44 41

FCC

'ADDA'

2673 F3FA 8B

FCB

\$8B

2674 F3FB 03

FCB

GEN

2675 F3FC 41 44 44 42

FCC

'ADDB'

2676 F400 CB

FCB

\$CB

2677 F401 03

FCB

GEN

2678 F402 41 44 44 44

FCC

'ADDD'

2679 F406 C3

FCB

\$C3

2680 F407 08

FCB

LIMM

2681 F408 41 4E 44 41

FCC

'ANDA'

2682 F40C 84

FCB

\$84

2683 F40D 03

FCB

GEN

2684 F40E 41 4E 44 42

FCC

'ANDB'

2685 F412 C4

FCB

\$C4

2686 F413 03

FCB

GEN

2687 F414 41 53 4C 20

FCC

'ASL'

2688 F418 68

FCB

\$68

2689 F419 04

FCB

GRP2

2690 F41A 41 53 4C 41

FCC

'ASLA'

2691 F41E 48

FCB

\$48

2692 F41F 01

FCB

INH

2693	F420	41 53 4C 42	FCC	'ASLB'
2694	F424	58	FCB	\$58
2695	F425	01	FCB	INH
2696	F426	41 53 4C 44	FCC	'ASLD'
2697	F42A	05	FCB	\$05
2698	F42B	01	FCB	INH
2699	F42C	41 53 52 20	FCC	'ASR'
2700	F430	67	FCB	\$67
2701	F431	04	FCB	GRP2
2702	F432	41 53 52 41	FCC	'ASRA'
2703	F436	47	FCB	\$47
2704	F437	01	FCB	INH
2705	F438	41 53 52 42	FCC	'ASRB'
2706	F43C	57	FCB	\$57
2707	F43D	01	FCB	INH
2708	F43E	42 43 43 20	FCC	'BCC'
2709	F442	24	FCB	\$24
2710	F443	05	FCB	REL
2711	F444	42 43 4C 52	FCC	'BCLR'
2712	F448	1D	FCB	\$1D
2713	F449	1F	FCB	SETCLR
2714	F44A	42 43 53 20	FCC	'BCS'
2715	F44E	25	FCB	\$25
2716	F44F	05	FCB	REL
2717	F450	42 45 51 20	FCC	'BEQ'
2718	F454	27	FCB	\$27
2719	F455	05	FCB	REL
2720	F456	42 47 45 20	FCC	'BGE'
2721	F45A	2C	FCB	\$2C
2722	F45B	05	FCB	REL
2723	F45C	42 47 54 20	FCC	'BGT'
2724	F460	2E	FCB	\$2E
2725	F461	05	FCB	REL
2726	F462	42 48 49 20	FCC	'BHI'
2727	F466	22	FCB	\$22
2728	F467	05	FCB	REL
2729	F468	42 48 53 20	FCC	'BHS'
2730	F46C	24	FCB	\$24
2731	F46D	05	FCB	REL
2732	F46E	42 49 54 41	FCC	'BITA'
2733	F472	85	FCB	\$85
2734	F473	03	FCB	GEN
2735	F474	42 49 54 42	FCC	'BITB'
2736	F478	C5	FCB	\$C5
2737	F479	03	FCB	GEN
2738	F47A	42 4C 45 20	FCC	'BLE'
2739	F47E	2F	FCB	\$2F
2740	F47F	05	FCB	REL
2741	F480	42 4C 4F 20	FCC	'BLO'
2742	F484	25	FCB	\$25
2743	F485	05	FCB	REL

2744	F486	42 4C 53 20	FCC	'BLS '	
2745	F48A	23	FCB	\$23	
2746	F48B	05	FCB	REL	
2747	F48C	42 4C 54 20	FCC	'BLT '	
2748	F490	2D	FCB	\$2D	
2749	F491	05	FCB	REL	
2750	F492	42 4D 49 20	FCC	'BMI '	
2751	F496	2B	FCB	\$2B	
2752	F497	05	FCB	REL	
2753	F498	42 4E 45 20	FCC	'BNE '	
2754	F49C	26	FCB	\$26	
2755	F49D	05	FCB	REL	
2756	F49E	42 50 4C 20	FCC	'BPL '	
2757	F4A2	2A	FCB	\$2A	
2758	F4A3	05	FCB	REL	
2759	F4A4	42 52 41 20	FCC	'BRA '	
2760	F4A8	20	FCB	\$20	
2761	F4A9	05	FCB	REL	
2762	F4AA	42 52 43 4C	FCC	'BRCL'	(BRCLR)
2763	F4AE	1F	FCB	\$1F	
2764	F4AF	13	FCB	BTB	
2765	F4B0	42 52 4E 20	FCC	'BRN '	
2766	F4B4	21	FCB	\$21	
2767	F4B5	05	FCB	REL	
2768	F4B6	42 52 53 45	FCC	'BRSE'	(BRSET)
2769	F4BA	1E	FCB	\$1E	
2770	F4BB	13	FCB	BTB	
2771	F4BC	42 53 45 54	FCC	'BSET'	
2772	F4C0	1C	FCB	\$1C	
2773	F4C1	14	FCB	SETCLR	
2774	F4C2	42 53 52 20	FCC	'BSR '	
2775	F4C6	8D	FCB	\$8D	
2776	F4C7	05	FCB	REL	
2777	F4C8	42 56 43 20	FCC	'BVC '	
2778	F4CC	28	FCB	\$28	
2779	F4CD	05	FCB	REL	
2780	F4CE	42 56 53 20	FCC	'BVS '	
2781	F4D2	29	FCB	\$29	
2782	F4D3	05	FCB	REL	
2783	F4D4	43 42 41 20	FCC	'CBA '	
2784	F4D8	11	FCB	\$11	
2785	F4D9	01	FCB	INH	
2786	F4DA	43 4C 42 20	FCC	'CLC '	
2787	F4DE	0C	FCB	\$0C	
2788	F4DF	01	FCB	INH	
2789	F4E0	43 4C 49 20	FCC	'CLI '	
2790	F4E4	0E	FCB	\$0E	
2791	F4E5	01	FCB	INH	
2792	F4E6	43 4C 52 20	FCC	'CLR '	
2793	F4EA	6F	FCB	\$6F	
2794	F4EB	04	FCB	GRP2	

2795 F4EC 43 4C 52 41
 2796 F4F0 4F
 2797 F4F1 01
 2798 F4F2 43 4C 52 42
 2799 F4F6 5F
 2800 F4F7 01
 2801 F4F8 43 4C 56 20
 2802 F4FC 0A
 2803 F4FD 01
 2804 F4FE 43 4D 50 41
 2805 F502 81
 2806 F503 03
 2807 F504 43 4D 50 42
 2808 F508 C1
 2809 F509 03
 2810 F50A 43 4F 4D 20
 2811 F50E 63
 2812 F50F 04
 2813 F510 43 4F 4D 41
 2814 F514 43
 2815 F515 01
 2816 F516 43 4F 4D 42
 2817 F51A 53
 2818 F51B 01
 2819 F51C 43 50 44 20
 2820 F520 83
 2821 F521 15
 2822 F522 43 50 58 20
 2823 F526 8C
 2824 F527 09
 2825 F528 43 50 59 20
 2826 F52C 8C
 2827 F52D 11
 2828 F52E 44 41 41 20
 2829 F532 19
 2830 F533 01
 2831 F534 44 45 43 20
 2832 F538 6A
 2833 F539 04
 2834 F53A 44 45 43 41
 2835 F53E 4A
 2836 F53F 01
 2837 F540 44 45 42 42
 2838 F544 5A
 2839 F545 01
 2840 F546 44 45 53 20
 2841 F54A 34
 2842 F54B 01
 2843 F54C 44 45 58 20
 2844 F550 09
 2845 F551 01

FCC 'CLRA'
 FCB \$4F
 FCB INH
 FCC 'CLRB'
 FCB \$5F
 FCB INH
 FCC 'CLV'
 FCB \$0A
 FCB INH
 FCC 'CMPA'
 FCB \$81
 FCB GEN
 FCC 'CMPB'
 FCB \$C1
 FCB GEN
 FCC 'COM'
 FCB \$63
 FCB GRP2
 FCC 'COMA'
 FCB \$43
 FCB INH
 FCC 'COMB'
 FCB \$53
 FCB INH
 FCC 'CPD'
 FCB \$83
 FCB CPD
 FCC 'CPX'
 FCB \$8C
 FCB XLIMM
 FCC 'CPY'
 FCB \$8C
 FCB YLIMM
 FCC 'DAA'
 FCB \$19
 FCB INH
 FCC 'DEC'
 FCB \$6A
 FCB GRP2
 FCC 'DECA'
 FCB \$4A
 FCB INH
 FCC 'DECB'
 FCB \$5A
 FCB INH
 FCC 'DES'
 FCB \$34
 FCB INH
 FCC 'DEX'
 FCB \$09
 FCB INH

2846	F552	44 45 59 20	FCC	'DEY '
2847	F556	09	FCB	\$09
2848	F557	02	FCB	P2INH
2849	F558	45 4F 52 41	FCC	'EORA'
2850	F55C	88	FCB	\$88
2851	F55D	03	FCB	GEN
2852	F55E	45 4F 52 42	FCC	'EORB'
2853	F562	C8	FCB	\$C8
2854	F563	03	FCB	GEN
2855	F564	46 44 49 56	FCC	'FDIV'
2856	F568	03	FCB	\$03
2857	F569	01	FCB	INH
2858	F56A	49 44 49 56	FCC	'IDIV'
2859	F56E	02	FCB	\$02
2860	F56F	01	FCB	INH
2861	F570	49 4E 43 20	FCC	'INC '
2862	F574	6C	FCB	\$6C
2863	F575	04	FCB	GRP2
2864	F576	49 4E 43 41	FCC	'INCA'
2865	F57A	4C	FCB	\$4C
2866	F57B	01	FCB	INH
2867	F57C	49 4E 43 42	FCC	'INCB'
2868	F580	5C	FCB	\$5C
2869	F581	01	FCB	INH
2870	F582	49 4E 53 20	FCC	'INS '
2871	F586	31	FCB	\$31
2872	F587	01	FCB	INH
2873	F588	49 4E 58 20	FCC	'INX '
2874	F58C	08	FCB	\$08
2875	F58D	01	FCB	INH
2876	F58E	49 4E 59 20	FCC	'INY '
2877	F592	08	FCB	\$08
2878	F593	02	FCB	P2INH
2879	F594	4A 4D 50 20	FCC	'JMP '
2880	F598	6E	FCB	\$6E
2881	F599	04	FCB	GRP2
2882	F59A	4A 53 52 20	FCC	'JSR '
2883	F59E	8D	FCB	\$8D
2884	F59F	07	FCB	NIMM
2885	F5A0	4C 44 41 41	FCC	'LDAA'
2886	F5A4	86	FCB	\$86
2887	F5A5	03	FCB	GEN
2888	F5A6	4C 44 41 42	FCC	'LDAB'
2889	F5AA	C6	FCB	\$C6
2890	F5AB	03	FCB	GEN
2891	F5AC	4C 44 44 20	FCC	'LDD '
2892	F5B0	CC	FCB	\$CC
2893	F5B1	08	FCB	LIMM
2894	F5B2	4C 44 53 20	FCC	'LDS '
2895	F5B6	8E	FCB	\$8E
2896	F5B7	08	FCB	LIMM

2897 F5B8 4C 44 58 20
 2898 F5BC CE
 2899 F5BD 09
 2900 F5BE 4C 44 59 20
 2901 F5C2 CE
 2902 F5C3 11
 2903 F5C4 4C 53 4C 20
 2904 F5C8 68
 2905 F5C9 04
 2906 F5CA 4C 53 4C 41
 2907 F5CE 48
 2908 F5CF 01
 2909 F5D0 4C 53 4C 42
 2910 F5D4 58
 2911 F5D5 01
 2912 F5D6 4C 53 4C 44
 2913 F5DA 05
 2914 F5DB 01
 2915 F5DC 4C 53 52 20
 2916 F5E0 64
 2917 F5E1 04
 2918 F5E2 4C 53 52 41
 2919 F5E6 44
 2920 F5E7 01
 2921 F5E8 4C 53 52 42
 2922 F5EC 54
 2923 F5ED 01
 2924 F5EE 4C 53 53 44
 2925 F5F2 04
 2926 F5F3 01
 2927 F5F4 4D 55 4C 20
 2928 F5F8 3D
 2929 F5F9 01
 2930 F5FA 4E 45 47 20
 2931 F5FE 60
 2932 F5FF 04
 2933 F600 4E 45 47 41
 2934 F604 40
 2935 F605 01
 2936 F606 4E 45 47 42
 2937 F60A 50
 2938 F60B 01
 2939 F60C 4E 4F 50 20
 2940 F610 01
 2941 F611 01
 2942 F612 4F 52 41 41
 2943 F616 8A
 2944 F617 03
 2945 F618 4F 52 41 42
 2946 F61C CA
 2947 F61D 03

FCC 'LDX'
 FCB \$CE
 FCB XLIMM
 FCC 'LDY'
 FCB \$CE
 FCB YLIMM
 FCC 'LSL'
 FCB \$68
 FCB GRP2
 FCC 'LSLA'
 FCB \$48
 FCB INH
 FCC 'LSLB'
 FCB \$58
 FCB INH
 FCC 'LSLD'
 FCB \$05
 FCB INH
 FCC 'LSR'
 FCB \$64
 FCB GRP2
 FCC 'LSRA'
 FCB \$44
 FCB INH
 FCC 'LSRB'
 FCB \$54
 FCB INH
 FCC 'LSRD'
 FCB \$04
 FCB INH
 FCC 'MUL'
 FCB \$3D
 FCB INH
 FCC 'NEG'
 FCB \$60
 FCB GRP2
 FCC 'NEGA'
 FCB \$40
 FCB INH
 FCC 'NEGB'
 FCB \$50
 FCB INH
 FCC 'NOP'
 FCB \$01
 FCB INH
 FCC 'ORAA'
 FCB \$8A
 FCB GEN
 FCC 'ORAB'
 FCB \$CA
 FCB GEN

2948 F61E 50 53 48 41
 2949 F622 36
 2950 F623 01
 2951 F624 50 53 48 42
 2952 F628 37
 2953 F629 01
 2954 F62A 50 53 48 58
 2955 F62E 3C
 2956 F62F 01
 2957 F630 50 53 48 59
 2958 F634 3C
 2959 F635 02
 2960 F636 50 55 4C 41
 2961 F63A 32
 2962 F63B 01
 2963 F63C 50 55 4C 42
 2964 F640 33
 2965 F641 01
 2966 F642 50 55 4C 58
 2967 F646 38
 2968 F647 01
 2969 F648 50 55 4C 59
 2970 F64C 38
 2971 F64D 02
 2972 F64E 52 4F 4C 20
 2973 F652 69
 2974 F653 04
 2975 F654 52 4F 4C 41
 2976 F658 49
 2977 F659 01
 2978 F65A 52 4F 4C 42
 2979 F65E 59
 2980 F65F 01
 2981 F660 52 4F 52 20
 2982 F664 66
 2983 F665 04
 2984 F666 52 4F 52 41
 2985 F66A 46
 2986 F66B 01
 2987 F66C 52 4F 52 42
 2988 F670 56
 2989 F671 01
 2990 F672 52 54 49 20
 2991 F676 3B
 2992 F677 01
 2993 F678 52 54 53 20
 2994 F67C 39
 2995 F67D 01
 2996 F67E 53 42 41 20
 2997 F682 10
 2998 F683 01

FCC 'PSHA'
 FCB \$36
 FCB INH
 FCC 'PSHB'
 FCB \$37
 FCB INH
 FCC 'PSHX'
 FCB \$3C
 FCB INH
 FCC 'PSHY'
 FCB \$3C
 FCB P2INH
 FCC 'PULA'
 FCB \$32
 FCB INH
 FCC 'PULB'
 FCB \$33
 FCB INH
 FCC 'PULX'
 FCB \$38
 FCB INH
 FCC 'PULY'
 FCB \$38
 FCB P2INH
 FCC 'ROL '
 FCB \$69
 FCB GRP2
 FCC 'ROLA'
 FCB \$49
 FCB INH
 FCC 'ROLB'
 FCB \$59
 FCB INH
 FCC 'ROR '
 FCB \$66
 FCB GRP2
 FCC 'RORA'
 FCB \$46
 FCB INH
 FCC 'RORB'
 FCB \$56
 FCB INH
 FCC 'RTI '
 FCB \$3B
 FCB INH
 FCC 'RTS '
 FCB \$39
 FCB INH
 FCC 'SBA '
 FCB \$10
 FCB INH

2999 F684 53 42 43 41
 3000 F688 82
 3001 F689 03
 3002 F68A 53 42 43 42
 3003 F68E C2
 3004 F68F 03
 3005 F690 53 45 43 20
 3006 F694 0D
 3007 F695 01
 3008 F696 53 45 49 20
 3009 F69A 0F
 3010 F69B 01
 3011 F69C 53 45 56 20
 3012 F6A0 0B
 3013 F6A1 01
 3014 F6A2 53 54 41 41
 3015 F6A6 87
 3016 F687 07
 3017 F6A8 53 54 41 42
 3018 F6AC C7
 3019 F6AD 07
 3020 F6AE 53 54 44 20
 3021 F6B2 CD
 3022 F6B3 07
 3023 F6B4 53 54 4F 50
 3024 F6B8 CF
 3025 F6B9 01
 3026 F6BA 53 54 53 20
 3027 F6BE BF
 3028 F6BF 07
 3029 F6C0 53 54 58 20
 3030 F6C4 CF
 3031 F6C5 10
 3032 F6C6 53 54 59 20
 3033 F6CA CF
 3034 F6CB 12
 3035 F6CC 53 55 42 41
 3036 F6D0 80
 3037 F6D1 03
 3038 F6D2 53 55 42 42
 3039 F6D6 C0
 3040 F6D7 03
 3041 F6D8 53 55 42 44
 3042 F6DC 83
 3043 F6DD 08
 3044 F6DE 53 57 49 20
 3045 F6E2 3F
 3046 F6E3 01
 3047 F6E4 54 41 42 20
 3048 F6E8 16
 3049 F6E9 01

FCC 'SBCA'
 FCB \$82
 FCB GEN
 FCC 'SBCB'
 FCB \$C2
 FCB GEN
 FCC 'SEC '
 FCB \$0D
 FCB INH
 FCC 'SEI '
 FCB \$0F
 FCB INH
 FCC 'SEV '
 FCB \$0B
 FCB INH
 FCC 'STAA'
 FCB \$87
 FCB NIMM
 FCC 'STAB'
 FCB \$C7
 FCB NIMM
 FCC 'STD '
 FCB \$CD
 FCB NIMM
 FCC 'STOP'
 FCB \$CF
 FCB INH
 FCC 'STS '
 FCB \$8F
 FCB NIMM
 FCC 'STX '
 FCB \$CF
 FCB XNIMM
 FCC 'STY '
 FCB \$CF
 FCB YNIMM
 FCC 'SUBA'
 FCB \$80
 FCB GEN
 FCC 'SUBB'
 FCB \$C0
 FCB GEN
 FCC 'SUBD'
 FCB \$83
 FCB LIMM
 FCC 'SWI '
 FCB \$3F
 FCB INH
 FCC 'TAB '
 FCB \$16
 FCB INH

```

3050 F6EA 54 41 50 20
3051 F6EE 06
3052 F6EF 01
3053 F6F0 54 42 41 20
3054 F6F4 17
3055 F6F5 01
3056 F6F6 54 50 41 20
3057 F6FA 07
3058 F6FB 01
3059 F6FC 54 45 53 54
3060 F700 00
3061 F701 01
3062 F702 54 53 54 20
3063 F706 6D
3064 F707 04
3065 F708 54 53 54 41
3066 F70C 4D
3067 F70D 01
3068 F70E 54 53 54 42
3069 F712 5D
3070 F713 01
3071 F714 54 53 58 20
3072 F718 30
3073 F719 01
3074 F71A 54 53 59 20
3075 F71E 30
3076 F71F 02
3077 F720 54 58 53 20
3078 F724 35
3079 F725 01
3080 F726 54 59 53 20
3081 F72A 35
3082 F72B 02
3083 F72C 57 41 49 20
3084 F730 3E
3085 F731 01
3086 F732 58 47 44 58
3087 F736 8F
3088 F737 01
3089 F738 58 47 44 59
3090 F73C 8F
3091 F73D 02
3092 F73E 42 52 53 45
3093 F742 12
3094 F743 16
3095 F744 42 52 43 4C
3096 F748 13
3097 F749 16
3098 F74A 42 53 45 54
3099 F74E 14

```

```

FCC 'TAP'
FCB $06
FCB INH
FCC 'TBA'
FCB $17
FCB INH
FCC 'TPA'
FCB $07
FCB INH
FCC 'TEST'
FCB $00
FCB INH
FCC 'TST'
FCB $6D
FCB GRP2
FCC 'TSTA'
FCB $4D
FCB INH
FCC 'TSTB'
FCB $5D
FCB INH
FCC 'TSX'
FCB $30
FCB INH
FCC 'TSY'
FCB $30
FCB P2INH
FCC 'TXS'
FCB $35
FCB INH
FCC 'TYS'
FCB $35
FCB P2INH
FCC 'WAI'
FCB $3E
FCB INH
FCC 'XGDX'
FCB $8F
FCB INH
FCC 'XGDY'
FCB $8F
FCB P2INH
FCC 'BRSE'
FCB $12
FCB BTBD
FCC 'BRCL'
FCB $13
FCB BTBD
FCC 'BSET'
FCB $14

```

bit direct modes for
disassembler.

3100	F74F	17		FCB	SETCLR	
3101	F750	42 43 4C 52		FCC	'BCLR'	
3102	F754	15		FCB	\$15	
3103	F755	17		FCB	SETCLR	
3104	F756	04		FCB	EOT	End of table
3105						
3106				*****		
3107	0000		PG1	EQU		\$0
3108	0001		PG2	EQU		\$1
3109	0002		PG3	EQU		\$2
3110	0003		PG4	EQU		\$3
3111						
3112				*****		
3113				*disassem() - disassemble the opcode.		
3114				*****		
3115						
3116	F757	DE 83	DISASSM	LDX	PC	address
3117	F759	A6 00		LDAA	0,X	opcode
3118	F75B	C6 00		LDAB	#PG1	
3119	F75D	81 18		CMPA	#\$18	
3120	F75F	27 0A		BEQ	DISP2	jump if page2
3121	F761	81 1A		CMPA	#\$1A	
3122	F763	27 05		BEQ	DISP3	jump if page3
3123	F765	81 CD		CMPA	#\$CD	
3124	F767	26 04		BNE	DISP1	jump if not page4
3125	F769	5C	DISP4	INCB		set up page value
3126	F76A	5C	DISP3	INCB		
3127	F76B	5C	DISP2	INCB		
3128	F76C	08		INX		
3129	F76D	DF 89	DISP1	STX	DISPC	point to opcode
3130	F76F	D7 94		STAB	PNORM	save page
3131	F771	A6 00		LDAA	0,X	get current opcode
3132	F773	97 87		STAA	BASEOP	
3133	F775	08		INX		
3134	F776	DF 89		STX	DISPC	point to next byte
3135	F778	81 5F		CMPA	#\$5F	
3136	F77A	23 0F		BLS	DIS1	jump if in range
3137	F77C	81 8D		CMPA	#\$8D	
3138	F77E	27 0B		BEQ	DIS1	jump if bsr
3139	F780	81 8F		CMPA	#\$8F	
3140	F782	27 07		BEQ	DIS1	jump if xgdx
3141	F784	81 CF		CMPA	#\$CF	
3142	F786	27 03		BEQ	DIS1	jump if stop
3143	F788	7E F8 22		JMP	DISGRP	try next part of map
3144	F78B	D6 94	DIS1	LDAB	PNORM	
3145	F78D	C1 02		CMPB	#PG3	
3146	F78F	25 04		BLO	DIS2	jump if page 1 or 2
3147	F791	BD F9 E2		JSR	DISILLOP	"illegal opcode"
3148	F794	39		RTS		
3149	F795	D6 87	DIS2	LDAB	BASEOP	opcode
3150	F797	5F		CLRB		class=null

3151	F798	BD F9 11		JSR	DISRCH	
3152	F79B	5D		TSTB		
3153	F79C	26 04		BNE	DISPEC	jump if opcode found
3154	F79E	BD F9 E2		JSR	DISILLOP	"illegal opcode"
3155	F7A1	39		RTS		
3156	F7A2	96 87	DISPEC	LDAA	BASEOP	
3157	F7A4	81 8D		CMPA	#\$8D	
3158	F7A6	26 04		BNE	DISPEC1	
3159	F7A8	C6 05		LDAB	#REL	
3160	F7AA	20 0A		BRA	DISPEC3	look for BSR opcode
3161	F7AC	81 8F	DISPEC1	CMPA	#\$8F	
3162	F7AE	27 04		BEQ	DISPEC2	jump if XGDX opcode
3163	F7B0	81 CF		CMPA	#\$CF	
3164	F7B2	26 05		BNE	DISINH	jump not STOP opcode
3165	F7B4	C6 01	DISPEC2	LDAB	#INH	
3166	F7B6	BD F9 11	DISPEC3	JSR	DISRCH	find other entry in table
3167	F7B9	D6 88	DISINH	LDAB	CLASS	
3168	F7BB	C1 01		CMPB	#INH	
3169	F7BD	26 18		BNE	DISREL	jump if not inherent
3170	F7BF	D6 94		LDAB	PNORM	
3171	F7C1	C1 00		CMPB	#PG1	
3172	F7C3	27 0E		BEQ	DISINH1	jump if page1
3173	F7C5	96 87		LDAA	BASEOP	get opcode
3174	F7C7	C6 02		LDAB	#P2INH	class=p2inh
3175	F7C9	BD F9 11		JSR	DISRCH	
3176	F7CC	5D		TSTB		
3177	F7CD	26 04		BNE	DISINH1	jump if found
3178	F7CF	BD F9 E2		JSR	DISILLOP	"illegal opcode"
3179	F7D2	39		RTS		
3180	F7D3	BD F9 34	DISINH1	JSR	PRNTMNE	
3181	F7D6	39		RTS		
3182	F7D7	D6 88	DISREL	LDAB	CLASS	
3183	F7D9	C1 05		CMPB	#REL	
3184	F7DB	26 10		BNE	DISBTD	
3185	F7DD	7D 00 94		TST	PNORM	
3186	F7E0	27 04		BEQ	DISREL1	jump if page1
3187	F7E2	BD F9 E2		JSR	DISILLOP	"illegal opcode"
3188	F7E5	39		RTS		
3189	F7E6	BD F9 34	DISREL1	JSR	PRNTMNE	output mnemonic
3190	F7E9	BD F9 6D		JSR	DISRELAD	compute relative address
3191	F7EC	39		RTS		
3192						
3193	F7ED	D6 88	DISBTD	LDAB	CLASS	
3194	F7EF	C1 17		CMPB	#SETCLRD	
3195	F7F1	27 04		BEQ	DISBTD1	
3196	F7F3	C1 16		CMPB	#BTBD	
3197	F7F5	26 11		BNE	DISBIT	jump not direct bitop
3198	F7F7	7D 00 94	DISBTD1	TST	PNORM	
3199	F7FA	27 04		BEQ	DISBTD2	jump if page 1
3200	F7FC	BD F9 E2		JSR	DISILLOP	
3201	F7FF	39		RTS		

3202	F800	BD F9 34	DISBTD2	JSR	PRNTMNE	
3203	F803	BD F9 C2		JSR	DISDIR	operand(direct)
3204	F806	20 06		BRA	DISBIT1	
3205	F808	BD F9 34	DISBIT	JSR	PRNTMNE	
3206	F80B	BD F9 51		JSR	DISINDX	operand(indexed)
3207	F80E	BD E4 21	DISBIT1	JSR	OUTSPAC	
3208	F811	BD F9 C2		JSR	DISDIR	mask
3209	F814	D6 88		LDAB	CLASS	
3210	F816	C1 13		CMPB	#BTB	
3211	F818	27 04		BEQ	DISBIT2	jump if btb
3212	F81A	C1 16		CMPB	#BTBD	
3213	F81C	26 03		BNE	DISBIT3	jump if not bit branch
3214	F81E	BD F9 6D	DISBIT2	JSR	DISRELAD	relative address
3215	F821	39	DISBIT3	RTS		
3216	F822	81 7F	DISGRP	CMPA	#\$7F	a=opcode
3217	F824	22 30		BHI	DISNEXT	try next part of map
3218	F826	D6 94		LDAB	PNORM	
3219	F828	C1 02		CMPB	#PG3	
3220	F82A	25 04		BLO	DISGRP2	jump if page 1 or 2
3221	F82C	BD F9 E2		JSR	DISILLOP	"illegal opcode"
3222	F82F	39		RTS		
3223	F830	84 6F	DISGRP2	ANDA	#\$6F	mask bit 4
3224	F832	5F		CLRB		class=null
3225	F833	BD F9 11		JSR	DISRCH	
3226	F836	5D		TSTB		
3227	F837	26 04		BNE	DISGRP3	jump if found
3228	F839	BD F9 E2		JSR	DISILLOP	"illegal opcode"
3229	F83C	39		RTS		
3230	F83D	BD F9 34	DISGRP3	JSR	PRNTMNE	
3231	F840	96 87		LDAA	BASEOP	get opcode
3232	F842	84 F0		ANDA	#\$F0	
3233	F844	81 60		CMPA	#\$60	
3234	F846	26 07		BNE	DISGRP4	jump if not 6x
3235	F848	BD F9 51		JSR	DISINDX	operand(indexed)
3236	F84B	BD E4 21		JSR	OUTSPAC	format
3237	F84E	39		RTS		
3238	F84F	BD F9 CF	DISGRP4	JSR	DISEXT	operand(extended)
3239	F852	BD E4 21		JSR	OUTSPAC	format
3240	F855	39		RTS		
3241	F856	81 87	DISNEXT	CMPA	#\$87	a=opcode
3242	F858	27 04		BEQ	DISNEX1	
3243	F85A	81 C7		CMPA	#\$C7	
3244	F85C	26 04		BNE	DISNEX2	
3245	F85E	BD F9 E2	DISNEX1	JSR	DISILLOP	"illegal opcode"
3246	F861	39		RTS		
3247	F862	84 CF	DISNEX2	ANDA	#\$CF	
3248	F864	5F		CLRB		class=null
3249	F865	BD F9 11		JSR	DISRCH	
3250	F868	5D		TSTB		
3251	F869	26 04		BNE	DISNEW	jump if mne found
3252	F86B	BD F9 E2		JSR	DISILLOP	"illegal opcode"

3253	F86E	39		RTS	
3254	F86F	96 87	DISNEW	LDAA BASEOP	
3255	F871	84 CF		ANDA #\$CF	
3256	F873	81 8D		CMPA #\$8D	
3257	F875	26 04		BNE DISNEW1	jump not jsr
3258	F877	C6 07		LDAB #NIMM	
3259	F879	20 16		BRA DISNEW4	
3260	F87B	81 8F	DISNEW1	CMPA #\$8F	
3261	F87D	26 04		BNE DISNEW2	jump not sts
3262	F87F	C6 07		LDAB #NIMM	
3263	F881	20 0E		BRA DISNEW4	
3264	F883	81 CF	DISNEW2	CMPA #\$CF	
3265	F885	26 04		BNE DISNEW3	jump not stx
3266	F887	C6 10		LDAB #XNIMM	
3267	F889	20 06		BRA DISNEW4	
3268	F88B	81 83	DISNEW3	CMPA #\$83	
3269	F88D	26 0C		BNE DISGEN	jump not subd
3270	F88F	C6 08		LDAB #LIMM	
3271	F891	BD F9 11	DISNEW4	JSR DISRCH	
3272	F894	5D		TSTB	
3273	F895	26 04		BNE DISGEN	jump if found
3274	F897	BD F9 E2		JSR DISILLOP	"illegal opcode"
3275	F89A	39		RTS	
3276	F89B	D6 88	DISGEN	LDAB CLASS	get class
3277	F89D	C1 03		CMPB #GEN	
3278	F89F	27 08		BEQ DISGEN1	
3279	F8A1	C1 07		CMPB #NIMM	
3280	F8A3	27 04		BEQ DISGEN1	
3281	F8A5	C1 08		CMPB #LIMM	
3282	F8A7	26 31		BNE DISXLN	jump if other class
3283	F8A9	96 87	DISGEN1	LDAA BASEOP	
3284	F8AB	84 CF		ANDA #\$CF	
3285	F8AD	81 83		CMPA #\$83	
3286	F8AF	26 0F		BNE DISGEN3	jump if not #\$83
3287	F8B1	D6 94		LDAB PNORM	
3288	F8B3	C1 02		CMPB #PG3	
3289	F8B5	25 09		BLO DISGEN3	jump not pg3 or 4
3290	F8B7	C6 15		LDAB #CPD	
3291	F8B9	BD F9 11		JSR DISRCH	look for cpd mne
3292	F8BC	C6 08		LDAB #LIMM	
3293	F8BE	D7 88		STAB CLASS	set class to limm
3294	F8C0	D6 94	DISGEN3	LDAB PNORM	
3295	F8C2	C1 01		CMPB #PG2	
3296	F8C4	27 04		BEQ DISGEN4	jump if page 2
3297	F8C6	C1 03		CMPB #PG4	
3298	F8C8	26 0C		BNE #DISGEN5	jump not page 2 or 4
3299	F8CA	96 87	DISGEN4	LDAA BASEOP	
3300	F8CC	84 B0		ANDA #\$B0	mask bits 6,3-0
3301	F8CE	81 A0		CMPA #\$A0	
3302	F8D0	27 04		BEQ DISGEN5	jump if \$Ax or \$Ex
3303	F8D2	BD F9 E2		JSR DISILLOP	"illegal opcode"

3304	F8D5	39		RTS	
3305	F8D6	BD F9 8C	DISGEN5	JSR DISGENRL	process general class
3306	F8D9	39		RTS	
3307	F8DA	D6 94	DISXLN	LDAB PNORM	
3308	F8DC	C1 01		CMPB #PG2	
3309	F8DE	27 04		BEQ DISXLN1	jump if page2
3310	F8E0	C1 02		CMPB #PG3	
3311	F8E2	26 13		BNE DISXLN4	jump not page3
3312	F8E4	96 87	DISXLN1	LDAA BASEOP	
3313	F8E6	84 CF		ANDA #SCF	
3314	F8E8	D6 88		LDAB CLASS	
3315	F8EA	C1 09		CMPB #XLIMM	
3316	F8EC	26 04		BNE DISXLN2	
3317	F8EE	C6 11		LDAB #YLIMM	
3318	F8F0	20 02		BRA DISXLN3	look for ylimm
3319	F8F2	C6 12	DISXLN2	LDAB #YNIMM	look for ynim
3320	F8F4	BD F9 11	DISXLN3	JSR DISRCH	
3321	F8F7	D6 94	DISXLN4	LDAB PNORM	
3322	F8F9	C1 02		CMPB #PG3	
3323	F8FB	25 0C		BLO DISXLN5	jump if page 1 or 2
3324	F8FD	96 87		LDAA BASEOP	get opcode
3325	F8FF	84 B0		ANDA #SB0	mask bits 6,3-0
3326	F901	81 A0		CMPA #SA0	
3327	F903	27 04		BEQ DISXLN5	jump opcode = \$Ax or SEx
3328	F905	BD F9 E2		JSR DISILLOP	"illegal opcode"
3329	F908	39		RTS	
3330	F909	C6 08	DISXLN5	LDAB #LIMM	
3331	F90B	D7 88		STAB CLASS	
3332	F90D	BD F9 8C		JSR DISGENRL	process general class
3333	F910	39		RTS	
3334					
3335					
3336					
3337					
3338					
3339					
3340					
3341					
3342	F911	CE F3 D8	DISRCH	LDX #MNETABL	point to top of table
3343	F914	A1 04	DISRCH1	CMPA 4,X	test opcode
3344	F916	26 0F		BNE DISRCH3	jump not this entry
3345	F918	5D		TSTB	
3346	F919	27 04		BEQ DISRCH2	jump if class=null
3347	F91B	E1 05		CMPB 5,X	test class
3348	F91D	26 08		BNE DISRCH3	jump not this entry
3349	F91F	E6 05	DISRCH2	LDAB 5,X	
3350	F921	D7 88		STAB CLASS	
3351	F923	DF 8D		STX MNEPTR	return ptr to mnemonic
3352	F925	5C		INCB	
3353	F926	39		RTS	return found
3354	F927	37	DISRCH3	PSHB	save class

*disrch(a=opcode,b=class)

*return b=0 if not found

* else mneptr=points to mnemonic

* class=class of opcode

```

3355 F928 C6 06          LDAB #6
3356 F92A 3A            ABX
3357 F92B E6 00          LDAB 0,X
3358 F92D C1 04          CMPB #EOT          test end of table
3359 F92F 33            PULB
3360 F930 26 E2          BNE DISRCH1
3361 F932 5F            CLRB
3362 F933 39            RTS          return not found
3363
3364
3365 *****
3366 *prntmne() - output the mnemonic pointed
3367 *at by mneptr.
3368 *****
3368 F934 DE 8D          PRNTMNE      LDX  MNEPTR
3369 F936 A6 00          LDAA 0,X
3370 F938 BD E1 76        JSR  OUTPUT          output char1
3371 F93B A6 01          LDAA 1,X
3372 F93D BD E1 76        JSR  OUTPUT          output char2
3373 F940 A6 02          LDAA 2,X
3374 F942 BD E1 76        JSR  OUTPUT          output char3
3375 F945 A6 03          LDAA 3,X
3376 F947 BD E1 76        JSR  OUTPUT          output char4
3377 F94A BD E4 21        JSR  OUTSPAC
3378 F94D BD E4 21        JSR  OUTSPAC
3379 F950 39            RTS
3380
3381 *****
3382 *disindx() - process indexed mode
3383 *****
3384
3385 F951 BD F9 C2          DISINDX      JSR  DISDIR          output $byte
3386 F954 86 2C          LDAA #','
3387 F956 BD E4 0B          JSR  OUTA          output ,
3388 F959 D6 94          LDAB PNORM
3389 F95B C1 01          CMPB #PG2
3390 F95D 27 04          BEQ  DISIND1          jump if page2
3391 F95F C1 03          CMPB #PG4
3392 F961 26 04          BNE  DISIND2          jump if not page4
3393 F963 86 59          DISIND1      LDAA #'Y'
3394 F965 20 02          BRA  DISIND3
3395 F967 86 58          DISIND2      LDAA #'X'
3396 F969 BD E4 0B          DISIND3      JSR  OUTA          output x or y
3397 F96C 39            RTS
3398
3399 *****
3400 *disrelad() - compute and output relative address.
3401 *****
3402
3403 F96D DE 89          DISRELAD      LDX  DISPC
3404 F96F E6 00          LDAB 0,X          get relative offset
3405 F971 08            INX

```

3406	F972	DF 89		STX	DISPC	
3407	F974	5D		TSTB		
3408	F975	2B 03		BMI	DISRLD1	jump if negative
3409	F977	3A		ABX		
3410	F978	20 04		BRA	DISRLD2	
3411	F97A	09	DISRLD1	DEX		
3412	F97B	5C		INCB		
3413	F97C	26 FC		BNE	DISRLD1	subtract
3414	F97E	DF 8B	DISRLD2	STX	BRADDR	save address
3415	F980	86 24		LDAA	#\$'	
3416	F982	BD E1 76		JSR	OUTPUT	
3417	F985	CE 00 8B		LDX	#BRADDR	
3418	F988	BD E4 1B		JSR	OUT2BSP	output address
3419	F98B	39		RTS		
3420						
3421						
3422						
3423						
3424						
3425						
3426	F98C	BD F9 34	DISGENRL	JSR	PRNTMNE	print mnemonic
3427	F98F	96 87		LDAA	BASEOP	get opcode
3428	F991	84 B0		ANDA	#\$B0	mask bits 6,3-0
3429	F993	81 80		CMPA	#\$80	
3430	F995	26 17		BNE	DISGRL2	jump if not immed
3431	F997	86 23		LDAA	##'	do immediate
3432	F999	BD E1 76		JSR	OUTPUT	
3433	F99C	BD F9 C2		JSR	DISDIR	
3434	F99F	D6 88		LDAB	CLASS	
3435	F9A1	C1 08		CMPB	#LIMM	
3436	F9A3	27 01		BEQ	DISGRL1	jump class = limm
3437	F9A5	39		RTS		
3438	F9A6	DE 89	DISGRL1	LDX	DISPC	
3439	F9A8	BD E4 0F		JSR	OUT1BYT	
3440	F9AB	DF 89		STX	DISPC	
3441	F9AD	39		RTS		
3442	F9AE	81 90	DISGRL2	CMPA	#\$90	
3443	F9B0	26 04		BNE	DISGRL3	jump not direct
3444	F9B2	BD F9 C2		JSR	DISDIR	do direct
3445	F9B5	39		RTS		
3446	F9B6	81 A0	DISGRL3	CMPA	#\$A0	
3447	F9B8	26 04		BNE	DISGRL4	jump not indexed
3448	F9BA	BD F9 51		JSR	DISINDX	do extended
3449	F9BD	39		RTS		
3450	F9BE	BD F9 CF	DISGRL4	JSR	DISEXT	do extended
3451	F9C1	39		RTS		
3452						
3453						
3454						
3455						
3456						

 *disgenrl() - output data for the general cases which
 *includes immediate, direct, indexed, and extended modes.

 *disdir() - output "\$ next byte"

```

3457 F9C2 86 24      DISDIR      LDAA #'$'
3458 F9C4 BD E4 0B      JSR      OUTA
3459 F9C7 DE 89      LDX      DISPC
3460 F9C9 BD E4 0F      JSR      OUT1BYT
3461 F9CC DF 89      STX      DISPC
3462 F9CE 39      RTS
3463
3464 *****
3465 *disext() - output "$ next 2 bytes"
3466 *****
3467
3468 F9CF      DISEXT      EQU      *
3469 F9CF 86 24      LDAA #'$'
3470 F9D1 BD E4 0B      JSR      OUTA
3471 F9D4 DE 89      LDX      DISPC
3472 F9D6 BD E4 1B      JSR      OUT2BSP
3473 F9D9 DF 89      STX      DISPC
3474 F9DB 39      RTS
3475
3476 *****
3477 *disillop() - output "illegal opcode"
3478 *****
3479
3480
3481 F9DC 49 4C 4C 4F DISMSG1      FCC      'ILLOP'
3482 F9E1 04      FCB      EOT
3483 F9E2      DISILLOP      EQU      *
3484 F9E2 3C      PSHX
3485 F9E3 CE F9 DC      LDX      #DISMSG1
3486 F9E6 BD E2 08      JSR      OUTSTRG
3487 F9E9 38      PULX
3488 F9EA 39      RTS
3489
3490      * redefine variable names
3490 0090      ROWCNT      EQU      PTR7+1
3491 0081      COLCNT      EQU      PTR0
3492 0082      CHARVAL      EQU      PTR0+1
3493 0093      SCANLINE      EQU      TMP3
3494
3495      * This routine scans the keyboard and displays the contents of
3496      * of CHARBUFF on a crt. The display is 5 rows of 6 characters.
3497      * The ASCII value of the depressed key is returned in register A.
3498      * If an abort key is depressed then control is passed to MAIN.
3499
3499 F9EB 37      VIDEO      PSHB      save regs this routine uses
3500 F9EC 3C      PSHX
3501 F9ED 18 3C      PSHY
3502
3503      * output the VSYNC pulse
3503 F9EF B6 10 08      VIDEO1      LDAA      PORTD
3504 F9F2 84 DF      ANDA      #SSLOW
3505 F9F4 B7 10 08      STAA      PORTD
3506 F9F7 CE 00 40      LDX      #64
3507 F9FA 09      VDELAY      DEX

```

3508	F9FB	26 FD		BNE	VDELAY	
3509	F9FD	8A 20		ORAA	#SSHIGH	
3510	F9FF	B7 10 08		STAA	PORTD	VSYNC pulse completed
3511				* READ THE KEYBOARD		
3512	FA02	18 CE 00 2C		LDY	#BLK LINES	blank line counter
3513	FA06	7D 00 00		TST	VALIDKEY	check for good key
3514	FA09	27 37		BEQ	MAIN1	no - then go to main1
3515				*check for no key or only shift key depressed (prevent repeating key)		
3516	FA0B	B6 10 29		LDAA	SPSR	
3517	FA0E	4F		CLRA		
3518	FA0F	B7 10 2A		STAA	SPDR	send good data; receive trash
3519	FA12	C6 0E		LDAB	#14	
3520	FA14	5A	DELAY10	DECB		
3521	FA15	26 FD		BNE	DELAY10	
3522	FA17	05		ASLD		kill time
3523	FA18	05		ASLD		kill time
3524	FA19	BD E2 63		JSR	HORPULSE	also loads shift reg.
3525	FA1C	18 09		DEY		decrement blank line count
3526	FA1E	B6 10 29		LDAA	SPSR	
3527	FA21	B7 10 2A		STAA	SPDR	send trash; receive good data
3528	FA24	3D		MUL		kill time for xmit
3529	FA25	05		ASLD		
3530	FA26	05		ASLD		
3531	FA27	B6 10 29		LDAA	SPSR	
3532	FA2A	B6 10 2A		LDAA	SPDR	get data
3533	FA2D	B1 FF		CMPA	#\$FF	is a key still depressed?
3534	FA2F	27 03		BEQ	NULL2	no- take branch
3535	FA31	7E FB 69		JMP	PREVID5	output frame, try next time
3536	FA34	7F 00 00	NULL2	CLR	VALIDKEY	
3537	FA37	05		ASLD		equalize
3538	FA38	01		NOP		equalize
3539	FA39	02		IDIV		kill time
3540	FA3A	05		ASLD		kill time
3541	FA3B	BD E2 63		JSR	HORPULSE	
3542	FA3E	18 09		DEY		decrement blank line count
3543	FA40	01		NOP		equalize
3544	FA41	01		NOP		equalize
3545	FA42	4F	MAIN1	CLRA		
3546	FA43	97 90		STAA	ROWCNT	
3547	FA45	97 81		STAA	COLCNT	
3548	FA47	86 FE		LDAA	#\$FE	set up (A) for first pattern
3549	FA49	F6 10 29		LDAB	SPSR	
3550	FA4C	B7 10 2A		STAA	SPDR	send first patt; rec garbage
3551	FA4F	0D		SEC		
3552	FA50	49		ROLA		set up (A) for next pattern
3553	FA51	C6 0C		LDAB	#12	
3554	FA53	5A	DELAY20	DECB		
3555	FA54	26 FD		BNE	DELAY20	
3556	FA56	01		NOP		
3557	FA57	01		NOP		

3558	FA58	BD E2 63		JSR	HORPULSE	
3559	FA5B	18 09		DEY		decrement blank line count
3560	FA5D	CE 00 0D		LDX	#13	
3561	FA60	09	DELAY30	DEX		
3562	FA61	26 FD		BNE	DELAY30	
3563	FA63	F6 10 29		LDAB	SPSR	
3564	FA66	B7 10 2A	KEY1	STAA	SPDR	sent good patt;rec good patt
3565	FA69	D6 81		LDAB	COLCNT	kill time
3566	FA6B	BD E2 63		JSR	HORPULSE	
3567	FA6E	18 09		DEY		decrement blank line count
3568	FA70	F6 10 29		LDAB	SPSR	
3569	FA73	F6 10 2A		LDAB	SPDR	get pattern
3570	FA76	C1 FF		CMPB	#\$FF	check for key depression
3571	FA78	26 42		BNE	DETERM2	yes-find which one
3572	FA7A	7C 00 81		INC	COLCNT	no-increment column count
3573	FA7D	0D		SEC		
3574	FA7E	49		ROLA		(A) has next pattern to send
3575	FA7F	24 09		BCC	LASTONE	all patterns have been sent
3576	FA81	C6 0A		LDAB	#10	
3577	FA83	5A	DELAY40	DECB		
3578	FA84	26 FD		BNE	DELAY40	
3579	FA86	01		NOP		kill time
3580	FA87	01		NOP		kill time
3581	FA88	20 DC		BRA	KEY1	send/receive next pattern
3582	FA8A	CE 00 0A	LASTONE	LDX	#10	
3583	FA8D	09	DELAY50	DEX		
3584	FA8E	26 FD		BNE	DELAY50	
3585	FA90	D6 81		LDAB	COLCNT	kill time
3586	FA92	BD E2 63		JSR	HORPULSE	
3587	FA95	18 09		DEY		decrement blank line count
3588	FA97	F7 10 2A		STAB	SPDR	send trash; get good pattern
3589	FA9A	3D		MUL		kill time
3590	FA9B	05		ASLD		
3591	FA9C	05		ASLD		
3592	FA9D	F6 10 29		LDAB	SPSR	
3593	FAA0	F6 10 2A		LDAB	SPDR	
3594	FAA3	C1 FF		CMPB	#\$FF	key depressed?
3595	FAA5	26 03		BNE	DETERM1	yes- take branch
3596	FAA7	7E FB 67		JMP	PREVID2	
3597	FAAA	D7 90	DETERM1	STAB	ROWCNT	save (B)
3598	FAAC	CE 00 08		LDX	#8	
3599	FAAF	09	DELAY60	DEX		
3600	FAB0	26 FD		BNE	DELAY60	
3601	FAB2	05		ASLD		kill time
3602	FAB3	01		NOP		
3603	FAB4	BD E2 63		JSR	HORPULSE	
3604	FAB7	18 09		DEY		decrement blank line count
3605	FAB9	3D		MUL		kill time for equalization
3606	FABA	D6 90		LDAB	ROWCNT	get row pattern
3607	FABC	D7 90	DETERM2	STAB	ROWCNT	save it again; equalized
3608	FABE	C6 0E		LDAB	#14	

3609	FAC0	5A	DELAY70	DECB		timing delay loop
3610	FAC1	26 FD		BNE	DELAY70	
3611	FAC3	01		NOP		kill time
3612	FAC4	01		NOP		
3613	FAC5	BD E2 63		JSR	HORPULSE	
3614	FAC8	18 09		DEY		
3615	FACA	4F		CLRA		
3616	FACB	D6 90		LDAB	ROWCNT	
3617	FACD	54	DETERM30	LSRB		
3618	FACE	24 03		BCC	DETERM20	
3619	FAD0	4C		INCA		
3620	FAD1	20 FA		BRA	DETERM30	
3621	FAD3	97 90	DETERM20	STAA	ROWCNT	
3622	FAD5	D6 90	DETERM40	LDAB	ROWCNT	kill time + equalize
3623	FAD7	4C		INCA		
3624	FAD8	81 08		CMPA	#MAXROWS	= 8
3625	FADA	26 F9		BNE	DETERM40	
3626	FADC	05		ASLD		kill time
3627	FADD	05		ASLD		kill time
3628	FADE	05		ASLD		kill time
3629	FADF	BD E2 63		JSR	HORPULSE	
3630	FAE2	18 09		DEY		
3631	FAE4	3D		MUL		
3632	FAE5	3D		MUL		
3633	FAE6	3D		MUL		
3634	FAE7	96 90		LDAA	ROWCNT	
3635	FAE9	48		ASLA		x2
3636	FAEA	48		ASLA		x4
3637	FAEB	48		ASLA		x8
3638	FAEC	9B 81		ADDA	COLCNT	
3639	FAEE	B1 B6 00		CMPA	SHIFTKEY	is it the shift key?
3640	FAF1	26 03		BNE	DUM1	
3641	FAF3	7E FB 6D		JMP	PREVID6	yes-go to display section
3642	FAF6	B1 B6 01	DUM1	CMPA	SHIFTLOW	if the key shifttable?
3643	FAF9	2D 3B		BLT	LOOKUP1	no
3644	FAFB	B1 B6 02		CMPA	SHIFTHI	is the key shifttable?
3645	FAFE	2E 3A		BGT	LOOKUP2	no
3646	FB00	F6 10 29		LDAB	SPSR	clear SPIF bit
3647	FB03	F6 B6 03		LDAB	SHFTCOLP	
3648	FB06	F7 10 2A		STAB	SPDR	send shift key pattern
3649	FB09	C6 03		LDAB	#3	
3650	FB0B	5A	DELAY90	DECB		timing loop
3651	FB0C	26 FD		BNE	DELAY90	
3652	FB0E	BD E2 63		JSR	HORPULSE	send HSYNC
3653	FB11	18 09		DEY		
3654	FB13	F6 10 29		LDAB	SPSR	clear SPIF
3655	FB16	F7 10 2A		STAB	SPDR	get row pattern
3656	FB19	18 3C		PSHY		kill time
3657	FB1B	18 38		PULY		kill time
3658	FB1D	36		PSHA		kill time
3659	FB1E	32		PULA		kill time

3660	FB1F	F6 10 29		LDAB	SPSR	
3661	FB22	F6 10 2A		LDAB	SPDR	
3662	FB25	F4 B6 04		ANDB	SHFTROWP	is the shift key down?
3663	FB28	27 03		BEQ	DUM2	yes-take branch
3664	FB2A	01		NOP		
3665	FB2B	20 05		BRA	DUM3	
3666	FB2D	BB B6 05	DUM2	ADDA	SHFTRNGE	offset for shift key
3667	FB30	D6 90		LDAB	ROWCNT	
3668	FB32	36	DUM3	PSHA		kill time
3669	FB33	32		PULA		kill time
3670	FB34	36		PSHA		kill time
3671	FB35	32		PULA		kill time
3672	FB36	D6 90	LOOKUP1	LDAB	ROWCNT	
3673	FB38	01		NOP		
3674	FB39	01		NOP		
3675	FB3A	91 82	LOOKUP2	CMPA	CHARVAL	match with last decoded key?
3676	FB3C	27 05		BEQ	FOUND	yes-take branch
3677	FB3E	97 82		STAA	CHARVAL	no-save as new CHARVAL
3678	FB40	7E FB 72		JMP	PREVID3	go display frame
3679	FB43	CE B6 06	FOUND	LDX	#TABLE	keyboard table
3680	FB46	16		TAB		
3681	FB47	3A		ABX		add offset
3682	FB48	A6 00		LDAA	0,X	get ASCII code
3683	FB4A	81 FF		CMPA	#\$FF	key implemented?
3684	FB4C	26 03		BNE	DUM4	yes-take branch
3685	FB4E	7E FB 75		JMP	PREVID10	no-continue display
3686	FB51	7C 00 00	DUM4	INC	VALIDKEY	indicate a valid key depressed
3687	FB54	18 38		PULY		restore registers
3688	FB56	38		PULX		
3689	FB57	33		PULB		
3690	FB58	81 18		CMPA	#KEYBREAK	check for abort from user
3691	FB5A	27 01		BEQ	RESET	
3692	FB5C	39		RTS		return to caller with code in A
3693	FB5D	7E E0 3E	RESET	JMP	MAIN	
3694	FB60	BD E2 63	HPULSE	JSR	HORPULSE	this section equalizes the
3695	FB63	3D		MUL		timing and outputs the
3696	FB64	3D		MUL		remaining blank lines prior
3697	FB65	3D		MUL		to the active display.
3698	FB66	3D		MUL		
3699	FB67	01	PREVID2	NOP		
3700	FB68	01		NOP		
3701	FB69	01	PREVID5	NOP		
3702	FB6A	05		ASLD		
3703	FB6B	05		ASLD		
3704	FB6C	01	PREVID4	NOP		
3705	FB6D	01	PREVID6	NOP		
3706	FB6E	05		ASLD		
3707	FB6F	01		NOP		
3708	FB70	01	PREVID7	NOP		
3709	FB71	3D		MUL		
3710	FB72	3D	PREVID3	MUL		

3711	FB73	01		NOP	
3712	FB74	01		NOP	
3713	FB75	01		PREVID10	NOP
3714	FB76	18 09		DEY	decrement counter
3715	FB78	26 E6		BNE HPULSE	do another blank line
3716	FB7A	BD E2 63		JSR HORPULSE	
3717			* initialize some variables		
3718	FB7D	86 03		LDAA #SLINES	
3719	FB7F	B7 00 2C		STAA >SLCNT	force extend for timing
3720	FB82	CE 00 27		LDX #DISPBUFF+2	
3721	FB85	DF 23		STX DISPTR	
3722			* toggle FIELD flag		
3723	FB87	7D 00 2B		TST FIELD	this section toggles the
3724	FB8A	27 05		BEQ TOGGLE1	FIELD variable
3725	FB8C	7F 00 2B		CLR FIELD	
3726	FB8F	20 04		BRA TOGGLE2	
3727	FB91	7C 00 2B	TOGGLE1	INC FIELD	
3728	FB94	05		ASLD	equalize time
3729	FB95	18 CE 00 04	TOGGLE2	LDY #CHARBUFF-1	
3730	FB99	D6 2B		LDAB FIELD	
3731	FB9B	18 3A		ABY	
3732	FB9D	18 DF 03		STY READBUFF	
3733	FBA0	CC 01 FF		LDD #\$01FF	
3734	FBA3	97 93		STAA SCANLINE	
3735	FBA5	D7 25		STAB DISPBUFF	set up first half of DISPBUFF
3736	FBA7	D7 26		STAB DISPBUFF+1	
3737	FBA9	D7 27		STAB DISPBUFF+2	
3738	FBAB	3D		MUL	kill time
3739	FBAC	3D		MUL	" "
3740	FBAD	3D		MUL	" "
3741	FBAE	3D		MUL	" "
3742			* DISPLAY DATA IN CHARACTER BUFFER		
3743	FBAF	B6 10 08	LINE1	LDAA PORTD	output HSYNC, update
3744	FBB2	84 DF		ANDA #SSLOW	index register Y, and ensure
3745	FBB4	B7 10 08		STAA PORTD	SPIF bit is cleared
3746	FBB7	F6 10 29		LDAB SPSR	
3747	FBBA	18 08		INY	
3748	FBBC	8A 20		ORAA #SSHIGH	
3749	FBBE	B7 10 08		STAA PORTD	
3750	FBC1	96 2B		LDAA FIELD	check FIELD for timing
3751	FBC3	27 06		BEQ EVEN1	
3752	FBC5	E6 00		LDAB 0,X	get pattern
3753	FBC7	DE 23		LDX DISPTR	get pointer
3754	FBC9	E7 00		STAB 0,X	put pattern at pointer address
3755	FBCB	18 E6 00	EVEN1	LDAB 0,Y	get ASCII code from buffer
3756	FBCE	08		INX	update DISPTR
3757	FBCF	DF 23		STX DISPTR	
3758	FBD1	96 25		LDAA DISPBUFF	send first pattern
3759	FBD3	B7 10 2A		STAA SPDR	
3760	FBD6	4F		CLRA	
3761	FBD7	58		ASLB	multiply ASCII code by 8

3762	FBD8 05		ASLD	
3763	FBD9 05		ASLD	
3764	FBDA 8F		XGDX	save 8*ASCII in X
3765	FBDB 96 26		LDAA DISPBUFF+1	send second pattern
3766	FBDD F6 10 29		LDAB SPSR	
3767	FBE0 B7 10 2A		STAA SPDR	
3768	FBE3 8F		XGDX	put 8*ASCII in D
3769	FBE4 DB 93		ADDB SCANLINE	add SCANLINE
3770	FBE6 C3 FC 5D		ADDD #FONTABLE-\$100	point at pattern
3771	FBE9 8F		XGDX	put pointer in X
3772	FBEA 96 27		LDAA DISPBUFF+2	send third pattern
3773	FBEC F6 10 29		LDAB SPSR	
3774	FBEF B7 10 2A		STAA SPDR	
3775	FBF2 7A 00 2C		DEC SLCNT	decrement SLCNT
3776	FBF5 27 0E		BEQ OUT1	done?
3777	FBF7 18 08		INY	increment CHARBUFF ptr
3778	FBF9 96 2B		LDAA FIELD	test FIELD
3779	FBFB 26 06		BNE EVEN2	
3780	FBFD E6 00		LDAB 0,X	get pattern
3781	FBFF DE 23		LDX DISPTR	get pointer
3782	FC01 E7 00		STAB 0,X	put pattern at pointer address
3783	FC03 20 AA	EVEN2	BRA LINE1	do next line
3784	FC05 96 2B	OUT1	LDAA FIELD	
3785	FC07 26 06		BNE EVEN3	
3786	FC09 E6 00		LDAB 0,X	get pattern
3787	FC0B DE 23		LDX DISPTR	get pointer
3788	FC0D E7 00		STAB 0,X	put pattern at pointer address
3789	FC0F 05	EVEN3	ASLD	kill time
3790	FC10 01		NOP	kill time
3791	FC11 01		NOP	kill time
3792	FC12 B6 10 08		LDAA PORTD	output HSYNC pulse and
3793	FC15 84 DF		ANDA #SSLOW	update variables
3794	FC17 B7 10 08		STAA PORTD	
3795	FC1A F6 10 29		LDAB SPSR	
3796	FC1D C6 03		LDAB #SLINES	
3797	FC1F 01		NOP	
3798	FC20 8A 20		ORAA #SSHIGH	
3799	FC22 B7 10 08		STAA PORTD	
3800	FC25 D7 2C		STAB SLCNT	
3801	FC27 18 DE 03		LDY READBUFF	point Y at start of char line
3802	FC2A 96 2B		LDAA FIELD	
3803	FC2C 27 06		BEQ TEST1	
3804	FC2E E6 00		LDAB 0,X	get pattern
3805	FC30 DE 23		LDX DISPTR	get pointer
3806	FC32 E7 00		STAB 0,X	put pattern at pointer address
3807	FC34 01	TEST1	NOP	kill time
3808	FC35 01		NOP	kill time
3809	FC36 96 25		LDAA DISPBUFF	send first pattern
3810	FC38 B7 10 2A		STAA SPDR	
3811	FC3B 7C 00 93		INC SCANLINE	increment SCANLINE count
3812	FC3E CE 00 24		LDX #DISPBUFF-1	

3813	FC41	DF 23		STX	DISPTR	update DISPTR
3814	FC43	96 26		LDAA	DISPBUFF+1	send second pattern
3815	FC45	F6 10 29		LDAB	SPSR	
3816	FC48	B7 10 2A		STAA	SPDR	
3817	FC4B	3D		MUL		kill time
3818	FC4C	05		ASLD		kill time
3819	FC4D	96 27		LDAA	DISPBUFF+2	send third pattern
3820	FC4F	F6 10 29		LDAB	SPSR	
3821	FC52	B7 10 2A		STAA	SPDR	
3822	FC55	3D		MUL		kill time
3823	FC56	05		ASLD		kill time
3824	FC57	05		ASLD		kill time
3825	FC58	96 2B		LDAA	FIELD	
3826	FC5A	26 02		BNE	LINE2	
3827	FC5C	3D		MUL		kill time
3828	FC5D	01		NOP		kill time
3829	FC5E	B6 10 08	LINE2	LDAA	PORTD	this section is exactly
3830	FC61	84 DF		ANDA	#SSLOW	the same as the previous
3831	FC63	B7 10 08		STAA	PORTD	section except the second
3832	FC66	F6 10 29		LDAB	SPSR	half of the display buffer
3833	FC69	18 08		INY		is displayed and first half
3834	FC6B	8A 20		ORAA	#SSHIGH	is filled with new patterns
3835	FC6D	B7 10 08		STAA	PORTD	
3836	FC70	96 2B		LDAA	FIELD	
3837	FC72	27 06		BEQ	EVEN11	
3838	FC74	E6 00		LDAB	0,X	
3839	FC76	DE 23		LDX	DISPTR	
3840	FC78	E7 00		STAB	0,X	
3841	FC7A	18 E6 00	EVEN11	LDAB	0,Y	
3842	FC7D	08		INX		
3843	FC7E	DF 23		STX	DISPTR	
3844	FC80	96 28		LDAA	DISPBUFF+3	
3845	FC82	B7 10 2A		STAA	SPDR	
3846	FC85	4F		CLRA		
3847	FC86	58		ASLB		
3848	FC87	05		ASLD		
3849	FC88	05		ASLD		
3850	FC89	8F		XGDX		
3851	FC8A	96 29		LDAA	DISPBUFF+4	
3852	FC8C	F6 10 29		LDAB	SPSR	
3853	FC8F	B7 10 2A		STAA	SPDR	
3854	FC92	8F		XGDX		
3855	FC93	DB 93		ADDB	SCANLINE	
3856	FC95	C3 FC 5D		ADDD	#FONTABLE-\$100	
3857	FC98	8F		XGDX		
3858	FC99	96 2A		LDAA	DISPBUFF+5	
3859	FC9B	F6 10 29		LDAB	SPSR	
3860	FC9E	B7 10 2A		STAA	SPDR	
3861	FCA1	7A 00 2C		DEC	SLCNT	
3862	FCA4	27 0E		BEQ	OUT2	
3863	FCA6	18 08		INY		

3864	FCA8	96 2B		LDAA FIELD	
3865	FCAA	26 06		BNE EVEN12	
3866	FCAC	E6 00		LDAB 0,X	
3867	FCAE	DE 23		LDX DISPTR	
3868	FCB0	E7 00		STAB 0,X	
3869	FCB2	20 AA	EVEN12	BRA LINE2	
3870	FCB4	96 2B	OUT2	LDAA FIELD	
3871	FCB6	26 06		BNE TEST2	
3872	FCB8	E6 00		LDAB 0,X	
3873	FCBA	DE 23		LDX DISPTR	
3874	FCBC	E7 00		STAB 0,X	
3875	FCBE	C6 03	TEST2	LDAB #SLINES	
3876	FCC0	D7 2C		STAB SLCNT	
3877	FCC2	01		NOP	
3878	FCC3	B6 10 08		LDAA PORTD	
3879	FCC6	84 DF		ANDA #SSLOW	
3880	FCC8	B7 10 08		STAA PORTD	
3881	FCCB	F6 10 29		LDAB SPSR	
3882	FCCE	18 08		INY	
3883	FCD0	8A 20		ORAA #SSHIGH	
3884	FCD2	B7 10 08		STAA PORTD	
3885	FCD5	96 2B		LDAA FIELD	
3886	FCD7	27 06		BEQ EVEN13	
3887	FCD9	E6 00		LDAB 0,X	
3888	FCDB	DE 23		LDX DISPTR	
3889	FCDD	E7 00		STAB 0,X	
3890	FCDF	CE 00 27	EVEN13	LDX #DISPBUFF+2	
3891	FCE2	DF 23		STX DISPTR	
3892	FCE4	D6 93		LDAB SCANLINE	
3893	FCE6	5C		INCB	
3894	FCE7	96 28		LDAA DISPBUFF+3	
3895	FCE9	B7 10 2A		STAA SPDR	
3896	FCEC	D7 93		STAB SCANLINE	
3897	FCEE	3D		MUL	
3898	FCEF	96 29		LDAA DISPBUFF+4	
3899	FCF1	F6 10 29		LDAB SPSR	
3900	FCF4	B7 10 2A		STAA SPDR	
3901	FCF7	96 93		LDAA SCANLINE	check to see if the character
3902	FCF9	81 09		CMPA #9	row is complete
3903	FCFB	26 1E		BNE NEXTSCLN	
3904	FCFD	86 01		LDAA #1	
3905	FCFF	97 93		STAA SCANLINE	
3906	FD01	96 2A		LDAA DISPBUFF+5	
3907	FD03	F6 10 29		LDAB SPSR	
3908	FD06	B7 10 2A		STAA SPDR	
3909	FD09	18 8C 00 22		CPY #CHARBUFF+BUFFLEN-1	check to see if
3910	FD0D	2C 22		BGE DONE	the entire CHARBUFF has
3911	FD0F	18 DF 03		STY READBUFF	been displayed
3912	FD12	96 2B		LDAA FIELD	
3913	FD14	26 02		BNE TEST20	

```

3914 FD16 3D          MUL
3915 FD17 01          NOP
3916 FD18 7E FB AF    TEST20    JMP    LINE1
3917 FD1B 18 DE 03    NEXTSCLN  LDY    READBUFF
3918 FD1E 96 2A        LDAA   DISPBUFF+5
3919 FD20 F6 10 29     LDAB   SPSR
3920 FD23 B7 10 2A     STAA   SPDR
3921 FD26 3D          MUL
3922 FD27 05          ASLD
3923 FD28 96 2B        LDAA   FIELD
3924 FD2A 26 02        BNE    TEST80
3925 FD2C 3D          MUL
3926 FD2D 01          NOP
3927 FD2E 7E FB AF    TEST80    JMP    LINE1
3928 FD31 18 CE 00 2EDONE  LDY    #TOTBLKS-BLKLINES  output the
3929 FD35 01          NOP                      remaining blank lines
3930 FD36 01          NOP                      to complete the display
3931 FD37 96 2B        LDAA   FIELD
3932 FD39 26 02        BNE    EVEN41
3933 FD3B 3D          MUL
3934 FD3C 01          NOP
3935 FD3D B6 10 08     EVEN41    LDAA   PORTD
3936 FD40 84 DF        ANDA   #SSLOW
3937 FD42 B7 10 08     STAA   PORTD
3938 FD45 05          ASLD
3939 FD46 05          ASLD
3940 FD47 01          NOP
3941 FD48 8A 20        ORAA   #SSHIGH
3942 FD4A B7 10 08     STAA   PORTD
3943 FD4D C6 13        LDAB   #19
3944 FD4F 5A          DELAY200  DECB
3945 FD50 26 FD        BNE    DELAY200
3946 FD52 05          ASLD
3947 FD53 18 09        DEY
3948 FD55 26 E6        BNE    EVEN41
3949 FD57 02          IDIV
3950 FD58 01          NOP
3951 FD59 05          ASLD
3952 FD5A 7E F9 EF     JMP    VIDEO1
3953
3954 *****
3955 * Font Table
3956 * contain the patterns necessary
3957 * for display of characters
3958 * 5 X 7 format
3959 *****
3960 FD5D FF FF        FONTABLE  FDB    $FFFF    ASCII space character
3961 FD5F FF FF        FDB    $FFFF
3962 FD61 FF FF        FDB    $FFFF
3963 FD63 FF FF        FDB    $FFFF
3964 FD65 FF CF        FDB    $FFCF    ASCII '!'

```

3965	FD67	CF CF	FDB	\$CFCF	
3966	FD69	CF FF	FDB	\$CFFF	
3967	FD6B	CF CF	FDB	\$CFCF	
3968	FD6D	FF D7	FDB	\$FFD7	ASCII '"'
3969	FD6F	D7 FF	FDB	\$D7FF	
3970	FD71	FF FF	FDB	\$FFFF	
3971	FD73	FF FF	FDB	\$FFFF	
3972	FD75	FF FF	FDB	\$FFFF	ASCII '#'
3973	FD77	D7 93	FDB	\$D793	
3974	FD79	FF 93	FDB	\$FF93	
3975	FD7B	D7 FF	FDB	\$D7FF	
3976	FD7D	FF C7	FDB	\$FFC7	ASCII '\$'
3977	FD7F	AB AF	FDB	\$ABAF	
3978	FD81	C7 EB	FDB	\$C7EB	
3979	FD83	AB C7	FDB	\$ABC7	
3980	FD85	FF 9F	FDB	\$FF9F	ASCII '%'
3981	FD87	9B F7	FDB	\$9BF7	
3982	FD89	EF DF	FDB	\$EFDF	
3983	FD8B	B3 F3	FDB	\$B3F3	
3984	FD8D	FF DF	FDB	\$FFDF	ASCII '&'
3985	FD8F	AF AF	FDB	\$AFAF	
3986	FD91	DF AB	FDB	\$DFAB	
3987	FD93	B7 CB	FDB	\$B7CB	
3988	FD95	FF F7	FDB	\$FFF7	ASCII '^'
3989	FD97	EF DF	FDB	\$EFDF	
3990	FD99	FF FF	FDB	\$FFFF	
3991	FD9B	FF FF	FDB	\$FFFF	
3992	FD9D	FF FB	FDB	\$FFFB	ASCII '('
3993	FD9F	F7 EF	FDB	\$F7EF	
3994	FDA1	EF EF	FDB	\$EFEF	
3995	FDA3	F7 FB	FDB	\$F7FB	
3996	FDA5	FF BF	FDB	\$FFBF	ASCII ')'
3997	FDA7	DF EF	FDB	\$DFEF	
3998	FDA9	EF EF	FDB	\$EFEF	
3999	FDAB	DF BF	FDB	\$DFBF	
4000	FDAD	FF FF	FDB	\$FFFF	ASCII '*'
4001	FDAF	EF C7	FDB	\$EFC7	
4002	FDB1	83 C7	FDB	\$83C7	
4003	FDB3	EF FF	FDB	\$EFFF	
4004	FDB5	FF FF	FDB	\$FFFF	ASCII '+'
4005	FDB7	EF EF	FDB	\$EFEF	
4006	FDB9	83 EF	FDB	\$83EF	
4007	FDBB	EF FF	FDB	\$EFFF	
4008	FDBD	FF FF	FDB	\$FFFF	ASCII ','
4009	FDBF	FF FF	FDB	\$FFFF	
4010	FDC1	CF CF	FDB	\$CFCF	
4011	FDC3	EF DF	FDB	\$EFDF	
4012	FDC5	FF FF	FDB	\$FFFF	ASCII ':'
4013	FDC7	FF FF	FDB	\$FFFF	
4014	FDC9	83 FF	FDB	\$83FF	
4015	FDCB	FF FF	FDB	\$FFFF	

4016	FDCD	FF FF	FDB	\$FFFF	ASCII 'V'
4017	FDCF	FF FF	FDB	\$FFFF	
4018	FDD1	FF FF	FDB	\$FFFF	
4019	FDD3	CF CF	FDB	\$CFCF	
4020	FDD5	FF FF	FDB	\$FFFF	ASCII 'Y'
4021	FDD7	FB F7	FDB	\$FBF7	
4022	FDD9	EF DF	FDB	\$EFDF	
4023	Fddb	BF FF	FDB	\$BFFF	
4024	FDDD	FF E7	FDB	\$FFE7	ASCII '0'
4025	FDDF	DB DB	FDB	\$DBDB	
4026	FDE1	DB DB	FDB	\$DBDB	
4027	FDE3	DB E7	FDB	\$DBE7	
4028	FDE5	FF EF	FDB	\$FFE7	ASCII '1'
4029	FDE7	CF EF	FDB	\$CFEF	
4030	FDE9	EF EF	FDB	\$EFEF	
4031	FDEB	EF C7	FDB	\$EFC7	
4032	FDED	FF C7	FDB	\$FFC7	ASCII '2'
4033	FDEF	BB FB	FDB	\$BBFB	
4034	FDF1	C7 FB	FDB	\$C7BF	
4035	FDF3	BF 83	FDB	\$BF83	
4036	FDF5	FF C7	FDB	\$FFC7	ASCII '3'
4037	FDF7	BB FB	FDB	\$BBFB	
4038	FDF9	C7 FB	FDB	\$C7FB	
4039	FDFB	BB C7	FDB	\$BBC7	
4040	FDFD	FF F7	FDB	\$FFF7	ASCII '4'
4041	FDFE	E7 D7	FDB	\$E7D7	
4042	FE01	B7 83	FDB	\$B783	
4043	FE03	F7 F7	FDB	\$F7F7	
4044	FE05	FF 83	FDB	\$FF83	ASCII '5'
4045	FE07	BF 87	FDB	\$BF87	
4046	FE09	FB FB	FDB	\$FBFB	
4047	FE0B	BB C7	FDB	\$BBC7	
4048	FE0D	FF E7	FDB	\$FFE7	ASCII '6'
4049	FE0F	DF BF	FDB	\$DFBF	
4050	FE11	87 BB	FDB	\$87BB	
4051	FE13	BB C7	FDB	\$BBC7	
4052	FE15	FF 83	FDB	\$FF83	ASCII '7'
4053	FE17	FB F7	FDB	\$FBF7	
4054	FE19	EF DB	FDB	\$EFDF	
4055	FE1B	DF DF	FDB	\$DFDF	
4056	FE1D	FF C7	FDB	\$FFC7	ASCII '8'
4057	FE1F	BB BB	FDB	\$BBBB	
4058	FE21	C7 BB	FDB	\$C7BB	
4059	FE23	BB C7	FDB	\$BBC7	
4060	FE25	FF C7	FDB	\$FFC7	ASCII '9'
4061	FE27	BB BB	FDB	\$BBBB	
4062	FE29	C3 FB	FDB	\$C3FB	
4063	FE2B	F7 CF	FDB	\$F7CF	
4064	FE2D	FF FF	FDB	\$FFFF	ASCII 'V'
4065	FE2F	CF CF	FDB	\$CFCF	
4066	FE31	FF CF	FDB	\$FFCF	

4067	FE33	CF FF	FDB	\$CFFF	
4068	FE35	FF CF	FDB	\$FFCF	ASCII ';'
4069	FE37	CF FF	FDB	\$CFFF	
4070	FE39	CF CF	FDB	\$CFCF	
4071	FE3B	EF DF	FDB	\$EFDf	
4072	FE3D	FF FB	FDB	\$FFFB	ASCII '<'
4073	FE3F	F7 EF	FDB	\$F7EF	
4074	FE41	DF EF	FDB	\$DFEF	
4075	FE43	F7 FB	FDB	\$F7FB	
4076	FE45	FF FF	FDB	\$FFFF	ASCII '='
4077	FE47	FF 83	FDB	\$FF83	
4078	FE49	FF 83	FDB	\$FF83	
4079	FE4B	FF FF	FDB	\$FFFF	
4080	FE4D	FF BF	FDB	\$FFBF	ASCII '>'
4081	FE4F	DF EF	FDB	\$DFEF	
4082	FE51	F7 EF	FDB	\$F7EF	
4083	FE53	DF BF	FDB	\$DFBF	
4084	FE55	FF C7	FDB	\$FFC7	ASCII '?'
4085	FE57	BB F7	FDB	\$BBF7	
4086	FE59	EF EF	FDB	\$EFEF	
4087	FE5B	FF EF	FDB	\$FFEf	
4088	FE5D	FF C7	FDB	\$FFC7	ASCII '@'
4089	FE5F	BB A3	FDB	\$BBA3	
4090	FE61	AB A3	FDB	\$ABA3	
4091	FE63	BF 83	FDB	\$BF83	
4092	FE65	FF C7	FDB	\$FFC7	ASCII 'A'
4093	FE67	BB BB	FDB	\$BBBB	
4094	FE69	83 BB	FDB	\$83BB	
4095	FE6B	BB BB	FDB	\$BBBB	
4096	FE6D	FF 87	FDB	\$FF87	ASCII 'B'
4097	FE6F	DB DB	FDB	\$DBDB	
4098	FE71	C7 DB	FDB	\$C7DB	
4099	FE73	DB 87	FDB	\$DB87	
4100	FE75	FF C7	FDB	\$FFC7	ASCII 'C'
4101	FE77	BB BF	FDB	\$BBBF	
4102	FE79	BF BF	FDB	\$BFBF	
4103	FE7B	BB C7	FDB	\$BBC7	
4104	FE7D	FF 87	FDB	\$FF87	ASCII 'D'
4105	FE7F	DB DB	FDB	\$DBDB	
4106	FE81	DB DB	FDB	\$DBDB	
4107	FE83	DB 87	FDB	\$DB87	
4108	FE85	FF 83	FDB	\$FF83	ASCII 'E'
4109	FE87	BF BF	FDB	\$BFBF	
4110	FE89	87 BF	FDB	\$87BF	
4111	FE8B	BF 83	FDB	\$BF83	
4112	FE8D	FF 83	FDB	\$FF83	ASCII 'F'
4113	FE8F	BF BF	FDB	\$BFBF	
4114	FE91	87 BF	FDB	\$87BF	
4115	FE93	BF BF	FDB	\$BFBF	
4116	FE95	FF C3	FDB	\$FFC3	ASCII 'G'
4117	FE97	BF BF	FDB	\$BFBF	

4118	FE99	A3 BB	FDB	\$A3BB	
4119	FE9B	BB C7	FDB	\$BBC7	
4120	FE9D	FF BB	FDB	\$FFBB	ASCII 'H'
4121	FE9F	BB BB	FDB	\$BBBB	
4122	FEA1	83 BB	FDB	\$83BB	
4123	FEA3	BB BB	FDB	\$BBBB	
4124	FEA5	FF C7	FDB	\$FFC7	ASCII 'I'
4125	FEA7	EF EF	FDB	\$EFEF	
4126	FEA9	EF EF	FDB	\$EFEF	
4127	FEAB	EF C7	FDB	\$EFC7	
4128	FEAD	FF FB	FDB	\$FFFB	ASCII 'J'
4129	FEAF	FB FB	FDB	\$FBFB	
4130	FEB1	FB FB	FDB	\$FBFB	
4131	FEB3	BB C7	FDB	\$BBC7	
4132	FEB5	FF BB	FDB	\$FFBB	ASCII 'K'
4133	FEB7	B7 AF	FDB	\$B7AF	
4134	FEB9	9F AF	FDB	\$9FAF	
4135	FEBB	B7 BB	FDB	\$B7BB	
4136	FEBD	FF BF	FDB	\$FFBF	ASCII 'L'
4137	FEBF	BF BF	FDB	\$BFBF	
4138	FEC1	BF BF	FDB	\$BFBF	
4139	FEC3	BF 83	FDB	\$BF83	
4140	FEC5	FF BB	FDB	\$FFBB	ASCII 'M'
4141	FEC7	93 AB	FDB	\$93AB	
4142	FEC9	AB BB	FDB	\$ABBB	
4143	FECB	BB BB	FDB	\$BBBB	
4144	FECD	FF BB	FDB	\$FFBB	ASCII 'N'
4145	FECF	9B AB	FDB	\$9BAB	
4146	FED1	B3 BB	FDB	\$B3BB	
4147	FED3	BB BB	FDB	\$BBBB	
4148	FED5	FF 83	FDB	\$FF83	ASCII 'O'
4149	FED7	BB BB	FDB	\$BBBB	
4150	FED9	BB BB	FDB	\$BBBB	
4151	FEDB	BB 83	FDB	\$BB83	
4152	FEDD	FF 87	FDB	\$FF87	ASCII 'P'
4153	FEDF	BB BB	FDB	\$BBBB	
4154	FEE1	87 BF	FDB	\$87BF	
4155	FEE3	BF BF	FDB	\$BFBF	
4156	FEE5	FF C7	FDB	\$FFC7	ASCII 'Q'
4157	FEE7	BB BB	FDB	\$BBBB	
4158	FEE9	BB AB	FDB	\$BBAB	
4159	FEED	B7 CB	FDB	\$B7CB	
4160	FEED	FF 87	FDB	\$FF87	ASCII 'R'
4161	FEED	BB BB	FDB	\$BBBB	
4162	FFF1	87 AF	FDB	\$87AF	
4163	FFF3	B7 BB	FDB	\$B7BB	
4164	FFF5	FF C7	FDB	\$FFC7	ASCII 'S'
4165	FFF7	BB DF	FDB	\$BBDF	
4166	FFF9	EF F7	FDB	\$EFF7	
4167	FFFB	BB C7	FDB	\$BBC7	
4168	FFFD	FF 83	FDB	\$FF83	ASCII 'T'

4169	FFFF	EF EF	FDB	\$EFEF	
4170	FF01	EF EF	FDB	\$EFEF	
4171	FF03	EF EF	FDB	\$EFEF	
4172	FF05	FF BB	FDB	\$FFBB	ASCII 'U'
4173	FF07	BB BB	FDB	\$BBBB	
4174	FF09	BB BB	FDB	\$BBBB	
4175	FF0B	BB C7	FDB	\$BBC7	
4176	FF0D	FF BB	FDB	\$FFBB	ASCII 'V'
4177	FF0F	BB BB	FDB	\$BBBB	
4178	FF11	D7 D7	FDB	\$D7D7	
4179	FF13	EF EF	FDB	\$EFEF	
4180	FF15	FF BB	FDB	\$FFBB	ASCII 'W'
4181	FF17	BB BB	FDB	\$BBBB	
4182	FF19	AB AB	FDB	\$ABAB	
4183	FF1B	AB D7	FDB	\$ABD7	
4184	FF1D	FF BB	FDB	\$FFBB	ASCII 'X'
4185	FF1F	BB D7	FDB	\$BBD7	
4186	FF21	EF D7	FDB	\$EFD7	
4187	FF23	BB BB	FDB	\$BBBB	
4188	FF25	FF BB	FDB	\$FFBB	ASCII 'Y'
4189	FF27	BB D7	FDB	\$BBD7	
4190	FF29	EF EF	FDB	\$EFEF	
4191	FF2B	EF EF	FDB	\$EFEF	
4192	FF2D	FF 83	FDB	\$FF83	ASCII 'Z'
4193	FF2F	FB F7	FDB	\$FBF7	
4194	FF31	EF DF	FDB	\$EFD7	
4195	FF33	BF 83	FDB	\$BF83	
4196	FF35	FF	FCB	\$FF	end of table

- * VECTOR TABLE- determine if user
- * has a specified service routine
- * for a particular interrupt.

4202					
4203	FF36	86 01	VECT1	LDAA #1	
4204	FF38	20 42		BRA VECTCHK	
4205	FF3A	82 02	VECT2	LDAA #2	
4206	FF3C	20 3E		BRA VECTCHK	
4207	FF3E	86 03	VECT3	LDAA #3	
4208	FF40	20 3A		BRA VECTCHK	
4209	FF42	86 04	VECT4	LDAA #4	
4210	FF44	20 36		BRA VECTCHK	
4211	FF46	86 05	VECT5	LDAA #5	
4212	FF48	20 32		BRA VECTCHK	
4213	FF4A	86 06	VECT6	LDAA #6	
4214	FF4C	20 2E		BRA VECTCHK	
4215	FF4E	86 07	VECT7	LDAA #7	
4216	FF50	20 2A		BRA VECTCHK	
4217	FF52	86 07	VECT8	LDAA #8	
4218	FF54	20 26		BRA VECTCHK	
4219	FF56	86 09	VECT9	LDAA #9	

4220	FF58	20 22		BRA	VECTCHK	
4221	FF5A	86 0A	VECT10	LDAA	#10	
4222	FF5C	20 1E		BRA	VECTCHK	
4223	FF5E	86 0B	VECT11	LDAA	#11	
4224	FF60	20 1A		BRA	VECTCHK	
4225	FF62	86 0C	VECT12	LDAA	#12	
4226	FF64	20 16		BRA	VECTCHK	
4227	FF66	86 0D	VECT13	LDAA	#13	
4228	FF68	20 12		BRA	VECTCHK	
4229	FF6A	86 0E	VECT14	LDAA	#14	
4230	FF6C	20 0E		BRA	VECTCHK	
4231	FF6E	86 0F	VECT15	LDAA	#15	
4232	FF70	20 0A		BRA	VECTCHK	
4233	FF72	86 10	VECT16	LDAA	#16	
4234	FF74	20 06		BRA	VECTCHK	
4235	FF76	86 11	VECT17	LDAA	#17	
4236	FF78	20 02		BRA	VECTCHK	
4237	FF7A	86 12	VECT18	LDAA	#18	
4238	FF7C	91 95	VECTCHK	CMPA	VECTR1	user routine for VECTR1?
4239	FF7E	26 04		BNE	VECTCHK1	
4240	FF80	DE 96		LDX	VJMP1	get address of routine
4241	FF82	6E 00		JMP	0,X	go there
4242	FF84	91 98	VECTCHK1	CMPA	VECTR2	check VECTR2
4243	FF86	26 04		BNE	VECTCHK2	
4244	FF88	DE 99		LDX	VJMP2	
4245	FF8A	6E 00		JMP	0,X	
4246	FF8C	3B	VECTCHK2	RTI		no service routine-return
4247						
4248	FFD6			ORG	\$FFD6	
4249			*** Vectors ***			
4250	FFD6	FF 36	VSCI	FDB	VECT1	
4251	FFD8	FF 3A	VSPI	FDB	VECT2	
4252	FFDA	FF 3E	VPAIE	FDB	VECT3	
4253	FFDC	FF 42	VPAO	FDB	VECT4	
4254	FFDE	FF 46	VTOF	FDB	VECT5	
4255	FFE0	00 9B	VTOC5	FDB	VOC5	dedicated oc5 vector
4256	FFE2	FF 4A	VTOC4	FDB	VECT6	
4257	FFE4	FF 4E	VTOC3	FDB	VECT7	
4258	FFE6	FF 52	VTOC2	FDB	VECT8	
4259	FFE8	FF 56	VTOC1	FDB	VECT9	
4260	FFEA	FF 5A	VTIC3	FDB	VECT10	
4261	FFEC	FF 5E	VTIC2	FDB	VECT11	
4262	FFEE	FF 62	VTIC1	FDB	VECT12	
4263	FFF0	FF 66	VRTI	FDB	VECT13	
4264	FFF2	FF 6A	VIRQ	FDB	VECT14	
4265	FFF4	FF 6E	VXIRQ	FDB	VECT15	
4266	FFF6	00 9E	VSWI	FDB	VSWIN	dedicated SWI vector
4267	FFF8	FF 72	VILLOP	FDB	VECT16	
4268	FFFA	FF 76	VCOP	FDB	VECT17	
4269	FFFC	FF 7A	VCLM	FDB	VECT18	
4270	FFFE	E0 00	VRST	FDB	PGMSTRT	

```

4271
4272
4273
4274
4275 B600
4276 B600 37
4277 B601 21
4278 B602 2F
4279 B603 7F
4280 B604 40
4281 B605 18
4282 B606 40 41 42 43
4283 B621 5E
4284 B622 0D
4285 B623 08
4286 B624 20
4287 B625 20
4288 B626 30 31 32 33
4289 B636 0D
4290 B637 FF
4291 B638 18
4292 B639 21 22 23 24
4293 B648

```

* KEYBOARD LOOKUP TABLE *

	ORG	\$B600	
SHIFTKEY	FCB	55	shiftkey matrix position
SHIFTLOW	FCB	33	first shiftable key
SHIFTHI	FCB	47	last shiftable key
SHFTCOLP	FCB	\$7F	column pattern for shift key
SHFTROWP	FCB	\$40	row pattern for shift key
SHFTRNGE	FCB	18	offset to shifted characters
TABLE	FCC	'@ABCDEFGHIJKLMNOPQRSTUVWXYZ'	
	FCB	\$5E	
	FCB	CRETURN	
	FCB	BACKSPCE	
	FCB	SPACE	
	FCB	SPACE	
	FCC	"0123456789:;,-./"	
	FCB	CRETURN	
	FCB	\$FF	
	FCB	CTLX	
	FCC	/'!#\$%&'()*+<=>?/'	
ENDKEYS	EQU	*	

BIBLIOGRAPHY

- [BAER80] Baer, J.-L., Computer Systems Architecture, Rockville, MD., Computer Science Press, 1980
- [GALL84] Gallup, M., "On-Chip Serial Interface Aids μ P System Expansion and Eases Traffic Crunch", Electronic Design, Vol. 32, May 3, 1984, pp. 355-364
- [HERR72] Herrick, C., Television Theory and Servicing, Reston, VA., Reston Publishing Co., 1972
- [KANE80] Kane, G., CRT Controller Handbook, Berkeley, CA., Osborne/McGraw Hill, 1980
- [LANC78] Lancaster, D., The Cheap Video Cookbook, Indianapolis, IN., Sams and Co., 1978
- [LIPO80] Lipovski, G., Microcomputer Interfacing: Principles and Practice, Lexington, MA., Lexington Books, 1980
- [MOT85A] Motorola Technical Data, MC68HC11A8 HCMOS Single-Chip Microcomputer, 1985
- [MOT85B] Motorola Technical Data, M68HC11EVM Evaluation Module User's Manual, 1985
- [TANN84] Tanenbaum, A., Structured Computer Organization, Englewood Cliffs, NJ. Prentice-Hall, 1984
- [ZAKS81] Zaks, R., From Chips to Systems, Berkeley, CA, Sybex, 1981

Vita

Kevin Eugene Williams was born in Canton, Illinois, on July 4, 1957, the son of Joyce Elenor Williams and Eddie Franklin Williams. In May 1979, he graduated from Kansas State University, Manhattan, Kansas, with a Bachelor of Science degree in Electrical Engineering. Upon graduation, he was commissioned as a Second Lieutenant in the United States Air Force. In January of 1981, he completed Undergraduate Pilot Training. He was assigned to RAF Lakenheath, United Kingdom, to fly the F-111F fighter-bomber until July, 1985. In August 1985, he entered The Graduate School of The University of Texas.

Permanent Address: 150 Martin Ave.
Canton, Illinois

This thesis was typed by Kevin E. Williams.

END

11-86

DT/C